

Initiation au Langage C



école supérieure de
génie informatique

Année scolaire 2024 - 2025

Kevin TRANCHO

Initiation au Langage C



école supérieure de
génie informatique

Cours de Kevin TRANCHO

Dispensé en Première et Seconde année
À l'ESGI Paris

Année scolaire 2024 - 2025

À propos

Ce support de cours accompagne les étudiants de l'ESGI Paris dans le cadre de leur initiation au Langage C (1^{ère} et 2^{ème} années).

Équipe pédagogique relative à ce cours

En cas de nécessité, les interlocuteurs relatifs à ce cours sont les suivants :

Kevin TRANCHO

ktrancho@myges.fr

Intervenant dans votre formation associé à ce cours.

Pauline ZEMOUR - Tuc-Luan TRAN

pzemour@esgi.fr - ttran@esgi.fr

Attachées de promotion alternance - janvier.

Frédéric SANANES

fsananes@esgi.fr

Directeur Pédagogique associé à ce cours.

Kevin TRANCHO

ktrancho@myges.fr

Responsable technique de ce support de cours.

Charles-David WAJNBERG - Kamal HENNOU

cwajnberg@esgi.fr - direction@esgi.fr

Direction pédagogique de l'ESGI.

Table des matières

I Modalités de ce cours	1
0 Préambule	5
0.1 Comment s'organise un cours ?	7
0.2 Quelques règles pour travailler ensemble	9
0.3 Des conseils pour réussir l'UE Langage C?	12
0.4 Comment s'organisent les évaluations ?	14
0.5 Planification cours et évaluations	19
0.6 Projet de fin de seconde année	23
II Découverte	29
1 Introduction	35
1.1 Informatique	35
1.2 Algorithmique	36
1.3 Langage C	37
1.4 Première application console	39
1.5 Préparer ses outils	41
1.6 Lire la documentation	52
1.7 Résumé	54
1.8 Entraînement	56
2 Variables	57
2.1 Introduction et motivation	57
2.2 Variables typées	59
2.3 Format d'affichage	63
2.4 Adresse d'une variable	69
2.5 Scanf	70
2.6 Résumé	73
2.7 Entraînement	76

TABLE DES MATIÈRES

3 Expressions	91
3.1 Opérations classiques	91
3.2 Division	94
3.3 Coercition : un changement de type	96
3.4 Réaffectation d'une variable avec opérateur	99
3.5 Résumé	102
3.6 Entraînement	104
4 Conditions	117
4.1 Sélection d'instructions avec if	117
4.2 Comparaisons	120
4.3 Opérateurs booléens	122
4.4 Ternaire : expression sous condition	127
4.5 Switch : se brancher un à résultat	128
4.6 Résumé	132
4.7 Entraînement	133
5 Boucles	157
5.1 While : répétition sous condition	157
5.2 Do while : répétition si besoin	158
5.3 For : un pas après l'autre	159
5.4 Contrôle de la boucle : break or continue	161
5.5 Résumé	164
5.6 Entraînement	165
III Notions de base	185
6 Fonctions	189
6.1 La fonction main	189
6.2 Définition de fonctions	191
6.3 Portée des variables	194
6.4 Déplacer sa définition de fonction	200
6.5 Résumé	202
6.6 Entraînement	203
7 Tableaux	237
7.1 Tableau à une dimension	237
7.2 Chaînes de caractères	241
7.3 Tableau à plusieurs dimensions	242
7.4 Résumé	246
7.5 Entraînement	247

TABLE DES MATIÈRES

8 Pointeurs	261
8.1 Un type d'adresse	261
8.2 Arithmétique des pointeurs	263
8.3 Allocation dynamique	266
8.4 Résumé	269
8.5 Entraînement	271
 IV Notions avancées	 297
9 Consolidation des bases	303
9.1 Remise dans le bain rapide	303
9.2 Entraînement	313
 10 Fichiers	 325
10.1 Ouverture et création de fichiers	325
10.2 Lecture et écriture	326
10.3 Se déplacer dans un fichier	331
10.4 Flux standards d'entrées et sorties	333
10.5 Résumé	335
10.6 Entraînement	337
 11 Structures	 343
11.1 Typedef	343
11.2 Structures	344
11.3 Définition	345
11.4 Avec des pointeurs	346
11.5 Unions	351
11.6 Énumérations	352
11.7 Résumé	355
11.8 Entraînement	356
 12 Programmation modulaire	 383
12.1 Retour sur la compilation	383
12.2 Directives préprocesseur	387
12.3 Makefiles	398
12.4 Résumé	403
12.5 Entraînement	405
 13 Types génériques	 419
13.1 Pointeurs de fonctions	420
13.2 Pointeurs génériques	426
13.3 Fonctions variadiques	432

TABLE DES MATIÈRES

13.4 Bibliothèques dynamiques	434
13.5 Résumé	440
13.6 Entraînement	442
14 Bonus : Opérations bit-à-bit	473
14.1 Décalages	473
14.2 Négation	474
14.3 Opérateurs booléens	475
14.4 Résumé	479
14.5 Entraînement	481
V Bonus : jouons rapidement avec SDL 1.2	487
15 Initialisation	493
15.1 Préambule	493
15.2 Mise en place	493
15.3 Fenêtre	500
16 Affichage	503
16.1 Dessiner un rectangle	504
16.2 Créer une surface	506
16.3 Images en BMP	510
16.4 Résumé	517
17 Événements	519
17.1 Récupération d'événements	519
17.2 Analyse d'événements	520
17.3 Résumé	526
18 Améliorations graphiques : SDL GFX	529
18.1 Dessiner des primitives	529
18.2 Écrire du texte	544
18.3 Faire tourner une image	546
18.4 Quelques usages de <code>SDL_imageFilter</code>	548
19 Texte : SDL TTF	555
19.1 Mise en place	555
19.2 Ouverture d'un Font	555
19.3 Générer une image depuis un texte	556
19.4 Effets de style	558

TABLE DES MATIÈRES

20 Musique : SDL Mixer	561
20.1 Mise en place	561
20.2 Jouer une musique	563
20.3 Canaux	565
21 Parallélisme : SDL Thread	569
21.1 Mise en place	569
21.2 Lancer un thread	569
21.3 Partager des ressources	570
21.4 Gestion parallèle d'événements, d'affichage et d'une simulation	573
22 Réseau : SDL Net	577
22.1 Mise en place	577
22.2 Avec TCP	577
22.3 Exemple d'utilisation pour un mini-jeux	580
VI Se préparer à l'examen	591
A Corrections exercices d'entraînement	595
A.1 Introduction	595
A.2 Variables	598
A.3 Expressions	611
A.4 Conditions	624
A.5 Boucles	652
A.6 Fonctions	673
A.7 Tableaux	718
A.8 Pointeurs	740
A.9 Consolidation des bases	770
A.10 Fichiers	795
A.11 Structures	807
A.12 Programmation modulaire	903
A.13 Types génériques	922
A.14 Opérations bit-à-bit	1020
B Examens des années précédentes	1029
B.1 Données statistiques partiels 2021 - 2023	1029
B.2 Sujets partiels semestre 1	1033
B.3 Sujets partiels semestre 2	1098

TABLE DES MATIÈRES

Exercices d’entraînement

Chapitre 1

1	Exercice – Formaliser une logique	36
2	Exercice – ★ Compiler Hello ESGI!	41
3	Exercice – ★ Votre Hello ESGI	56

Chapitre 2

4	Exercice – ★ Déclarer et définir des variables	62
5	Exercice – ★★ Afficher des variables	69
6	Exercice – ★★ Lire et afficher une variable	72
7	Exercice – ★★ Affichage formaté hexadécimal	76
8	Exercice – ★ Affectations	76
9	Exercice – ★ Traduction en Langage C	77
10	Exercice – ★★ Dépassement capacité d’un int	78
11	Exercice – ★★ Imprécision flottante	79
12	Exercice – ★★★ Partie entière d’un flottant	80
13	Exercice – ★★★ Hexadécimal ?	81
14	Exercice – ★★★★ Un message secret	82
15	Exercice – ∞ Mini-projet : dessins de formes	83

Chapitre 3

16	Exercice – ★ Calculer pour l’utilisateur	93
17	Exercice – ★★ Opérations	104
18	Exercice – ★★ Division par zéro	105
19	Exercice – ★★ Affection d’une addition ?	106
20	Exercice – ★★ Imprécision et opérations	107
21	Exercice – ★★★ Message codé	108
22	Exercice – ★★ Afficher sinus et cosinus d’un angle	109
23	Exercice – ★★★ Conversion d’une adresse IP	109
24	Exercice – ★★★★ Suite de Fibonacci par la formule de Binet	110
25	Exercice – ∞ Shooter	110

Chapitre 4

26	Exercice – ★ Déterminer une catégorie d’âge	122
27	Exercice – ★ Intersection d’événements	123
28	Exercice – ★★ Calculer l’amende d’un excès de vitesse	126
29	Exercice – ★★ Acheter un article	133

EXERCICES D'ENTRAÎNEMENT

30	Exercice – ★ Menu et addition	134
31	Exercice – ★★★ Calculatrice	135
32	Exercice – ★ Racines d'un polynôme du second degré	136
33	Exercice – ★★★ Code obscurantiste	137
34	Exercice – ★★★ Validation d'un mot de passe	138
35	Exercice – ★★★ Calcul de l'impôt sur le revenu	139
36	Exercice – ∞ Lancé de rayon sur sphères	141
37	Exercice – ∞ Saut d'un personnage et suivi par caméra	148

Chapitre 5

38	Exercice – ★ Liste des diviseurs	165
39	Exercice – ★★★ Force brute	165
40	Exercice – ★★★★ Affichage en binaire	165
41	Exercice – ★★★ PGCD	165
42	Exercice – ★ Exponentiation d'un entier	166
43	Exercice – ★★★ Jeu du plus ou moins	167
44	Exercice – ★★★ Implémentation racine carré	168
45	Exercice – ∞ Dessin de fractales	170

Chapitre 6

46	Exercice – ★ Définir une fonction	193
47	Exercice – ★★ Fonction de menu	203
48	Exercice – ★★ Calcul moyenne	204
49	Exercice – ★★ Utilisation variables locales	205
50	Exercice – ★★★ Compléter fonctions	206
51	Exercice – ★★★ Calculatrice avec mémoire : séparation en fonctions	207
52	Exercice – ★★★★ Exponentiation et performances	209
53	Exercice – ★★★★ Affrontement de personnages	210
54	Exercice – ★★★ Racine carré par dichotomie	211
55	Exercice – ★★★ Calcul coefficients binomiaux	213
56	Exercice – ★★★ Développement de Taylor : calcul de sinus	214
57	Exercice – ∞ Contrôle de l'aléatoire	217
58	Exercice – ∞ Courbes récursives	223

Chapitre 7

59	Exercice – ★★ Création et affectation	247
60	Exercice – ★★ Lire valeurs	248
61	Exercice – ★★ Statistiques sur un tableau	249
62	Exercice – ★★ Coder les fonction de string.h	250
63	Exercice – ★★★ Compter les occurrences de lettres	251
64	Exercice – ★★★ Décodage Vigenère	252
65	Exercice – ★★★ Gestion d'une liste d'entiers	252

EXERCICES D'ENTRAÎNEMENT

66	Exercice – ★★ Dichotomie dans un tableau	254
67	Exercice – ★★★ Suite de Fibonacci	256
68	Exercice – ★★★ Chiffrement par permutation d'alphabet	258

Chapitre 8

69	Exercice – ★★★ Fonction pour échanger des valeurs	262
70	Exercice – ★ Pointeurs sur des variables	271
71	Exercice – ★★ Pointinception	272
72	Exercice – ★★★ Construction liste extensible	273
73	Exercice – ★★ Échanger dans un tableau	275
74	Exercice – ★★★ Tableau de variables	276
75	Exercice – ★★ Adresse sur un entier	277
76	Exercice – ★★ Dupliquer chaîne de caractère via arithmétique de pointeurs	277
77	Exercice – ★★ Visualiser les sauts d'octets et interprétation des valeurs	278
78	Exercice – ★★ Concept de pointeur	280
79	Exercice – ★★★ Concaténation de chaînes de caractères	282
80	Exercice – ★★★ Gestion d'une liste de noms	283
81	Exercice – ★★★ Tableau dynamique à deux dimensions	284
82	Exercice – ∞ Labyrinthe	286

Chapitre 9

83	Exercice – ★★ Lecture d'arguments en ligne de commande	313
84	Exercice – ★★★ Options depuis ligne de commande	313
85	Exercice – ★ Allouer dynamiquement une variable	314
86	Exercice – ★★ Allouer dynamiquement tableau	314
87	Exercice – ★★ Allocation dynamique deux dimensions naïve	315
88	Exercice – ★★ Allocation dynamique deux dimensions en deux allocations	316
89	Exercice – ★★ Allocation dynamique via une fonction	316
90	Exercice – ★★★ Statistiques sur liste d'entiers	317
91	Exercice – ★★★ Extraire de l'information formatée d'une chaîne de caractères	318
92	Exercice – ★★★ Mini-interpréteur sur liste d'entiers	319
93	Exercice – ★★★ Jeu du Morpion	320
94	Exercice – ★★★★ Enregistrement et recherche de numéros	321
95	Exercice – ★★★★ Suite de Fibonacci généralisée et optimisée	322

Chapitre 10

96	Exercice – ★ Création d'un fichier	337
97	Exercice – ★★ Lire des fichiers	337
98	Exercice – ★★ Fichier en binaire	338

EXERCICES D'ENTRAÎNEMENT

99	Exercice – ★★ Éditer un fichier	338
100	Exercice – ★★ Compteur de lancements	339
101	Exercice – ★★★ Codage Vigenère depuis fichier	339
102	Exercice – ★★★ Sauvegarde et chargement en binaire	340
103	Exercice – ★★★★ Enregistrement et recherche de numéros avec sauvegarde	341
 Chapitre 11		
104	Exercice – ★ Créer une structure	356
105	Exercice – ★★ Passage par copie	357
106	Exercice – ★★ Passage par adresse	358
107	Exercice – ★★ Manipulation dynamique	359
108	Exercice – ★★★ Définir une structure Vecteur2d	360
109	Exercice – ★★★ Structure pour gérer une grille	361
110	Exercice – ★★★★ Gérer un combat de personnages	363
111	Exercice – ★★★ Implémenter une liste chaînée	364
112	Exercice – ∞ Algorithmes génétiques et problème du voyageur de commerce	366
113	Exercice – ∞ Combat de créatures par fichiers	372
 Chapitre 12		
114	Exercice – ★★ Mise en place d'un Makefile	405
115	Exercice – ★★★★ Profilage d'un code	407
116	Exercice – ★★ Listes chaînées et ajouts	409
117	Exercice – ★★★★ Implémentation table de hachage	411
118	Exercice – ★★★★ Représentation d'un lexique	416
 Chapitre 13		
119	Exercice – ★★★ Trier une liste de points	442
120	Exercice – ★★★★ Benchmark de qsort	442
121	Exercice – ★★★★ Trier des pointeurs de fonctions	443
122	Exercice – ★★★ Changer de langue	445
123	Exercice – ★★★★★ Simulation de combattants génériques	447
124	Exercice – ★★★★★ HashMap et ArrayList génériques	451
125	Exercice – ★★★★★ Tableaux dynamiques à n dimensions	452
126	Exercice – ∞ Algorithme d'exponentiation générique	455
127	Exercice – ∞ Space invaders modulaire	458
 Chapitre 14		
128	Exercice – ★ Application opérations bit-à-bit	481
129	Exercice – ★★★ Gérer une grille sur un entier	483
130	Exercice – ★★★ Buffer pour lecture et écriture bit-à-bit	484

Corrections des exercices

Chapitre A

1	Correction – Formaliser une logique	595
2	Correction – ★ Compiler Hello ESGI!	596
3	Correction – ★ Votre Hello ESGI	596
4	Correction – ★ Déclarer et définir des variables	598
5	Correction – ★★ Afficher des variables	598
6	Correction – ★★ Lire et afficher une variable	598
7	Correction – ★★ Affichage formaté hexadécimal	599
8	Correction – ★ Affectations	599
9	Correction – ★ Traduction en Langage C	600
10	Correction – ★★ Dépassement capacité d'un int	601
11	Correction – ★★ Imprécision flottante	602
12	Correction – ★★★ Partie entière d'un flottant	603
13	Correction – ★★★★ Hexadécimal ?	603
14	Correction – ★★★★ Un message secret	604
15	Correction – ∞ Mini-projet : dessins de formes	605
16	Correction – ★ Calculer pour l'utilisateur	611
17	Correction – ★★ Opérations	611
18	Correction – ★★ Division par zéro	612
19	Correction – ★★ Affection d'une addition ?	613
20	Correction – ★★ Imprécision et opérations	614
21	Correction – ★★ Message codé	614
22	Correction – ★★ Afficher sinus et cosinus d'un angle	615
23	Correction – ★★★ Conversion d'une adresse IP	615
24	Correction – ★★★★ Suite de Fibonacci par la formule de Binet	616
25	Correction – ∞ Shooter	617
26	Correction – ★ Déterminer une catégorie d'âge	624
27	Correction – ★ Intersection d'événements	624
28	Correction – ★★ Calculer l'amende d'un excès de vitesse	625
29	Correction – ★★ Acheter un article	626
30	Correction – ★★ Menu et addition	627
31	Correction – ★★★ Calculatrice	628
32	Correction – ★★ Racines d'un polynôme du second degré	630
33	Correction – ★★★ Code obscurantiste	632
34	Correction – ★★★ Validation d'un mot de passe	633
35	Correction – ★★★ Calcul de l'impôt sur le revenu	634

CORRECTIONS DES EXERCICES

36	Correction – ∞ Lancé de rayon sur sphères	636
37	Correction – ∞ Saut d'un personnage et suivi par caméra	642
38	Correction – ★★ Liste des diviseurs	652
39	Correction – ★★★ Force brute	653
40	Correction – ★★★★ Affichage en binaire	654
41	Correction – ★★★ PGCD	656
42	Correction – ★★ Exponentiation d'un entier	657
43	Correction – ★★★ Jeu du plus ou moins	658
44	Correction – ★★★ Implémentation racine carré	659
45	Correction – ∞ Dessin de fractales	660
46	Correction – ★ Définir une fonction	673
47	Correction – ★★ Fonction de menu	673
48	Correction – ★★ Calcul moyenne	674
49	Correction – ★★ Utilisation variables locales	676
50	Correction – ★★★ Compléter fonctions	678
51	Correction – ★★★ Calculatrice avec mémoire : séparation en fonctions	681
52	Correction – ★★★★ Exponentiation et performances	684
53	Correction – ★★★★ Affrontement de personnages	686
54	Correction – ★★★ Racine carré par dichotomie	691
55	Correction – ★★★ Calcul coefficients binomiaux	692
56	Correction – ★★★ Développement de Taylor : calcul de sinus	693
57	Correction – ∞ Contrôle de l'aléatoire	696
58	Correction – ∞ Courbes récursives	702
59	Correction – ★★ Création et affectation	718
60	Correction – ★★ Lire valeurs	718
61	Correction – ★★ Statistiques sur un tableau	719
62	Correction – ★★ Coder les fonction de string.h	721
63	Correction – ★★★ Compter les occurrences de lettres	723
64	Correction – ★★★ Décodage Vigenère	724
65	Correction – ★★★ Gestion d'une liste d'entiers	726
66	Correction – ★★★ Dichotomie dans un tableau	730
67	Correction – ★★★★ Suite de Fibonacci	732
68	Correction – ★★★★ OOO Chiffrement par permutation d'alphabet	737
69	Correction – ★★★ Fonction pour échanger des valeurs	740
70	Correction – ★ Pointeurs sur des variables	740
71	Correction – ★★ Pointinception	741
72	Correction – ★★★ Construction liste extensible	741
73	Correction – ★★ Échanger dans un tableau	744
74	Correction – ★★★ Tableau de variables	745
75	Correction – ★★ Adresse sur un entier	746
76	Correction – ★★ Dupliquer chaîne de caractère via arithmétique de pointeurs	746

CORRECTIONS DES EXERCICES

77	Correction – ★★ Visualiser les sauts d'octets et interprétation des valeurs	747
78	Correction – ★★ Concept de pointeur	748
79	Correction – ★★★ Concaténation de chaînes de caractères	749
80	Correction – ★★★ Gestion d'une liste de noms	750
81	Correction – ★★★★ Tableau dynamique à deux dimensions	753
82	Correction – ∞ Labyrinthe	755
83	Correction – ★★ Lecture d'arguments en ligne de commande	770
84	Correction – ★★★ Options depuis ligne de commande	770
85	Correction – ★ Allouer dynamiquement une variable	771
86	Correction – ★★ Allouer dynamiquement tableau	772
87	Correction – ★★ Allocation dynamique deux dimensions naïve	773
88	Correction – ★★ Allocation dynamique deux dimensions en deux allocations	774
89	Correction – ★★ Allocation dynamique via une fonction	775
90	Correction – ★★★ Statistiques sur liste d'entiers	776
91	Correction – ★★★ Extraire de l'information formatée d'une chaîne de caractères	778
92	Correction – ★★★ Mini-interpréteur sur liste d'entiers	779
93	Correction – ★★★ Jeu du Morpion	780
94	Correction – ★★★★ Enregistrement et recherche de numéros	785
95	Correction – ★★★★★ Suite de Fibonacci généralisée et optimisée	791
96	Correction – ★ Création d'un fichier	795
97	Correction – ★★ Lire des fichiers	795
98	Correction – ★★ Fichier en binaire	796
99	Correction – ★★ Éditer un fichier	796
100	Correction – ★★ Compteur de lancements	797
101	Correction – ★★★ Codage Vigenère depuis fichier	798
102	Correction – ★★★ Sauvegarde et chargement en binaire	800
103	Correction – ★★★★ Enregistrement et recherche de numéros avec sauvegarde	803
104	Correction – ★ Créer une structure	807
105	Correction – ★★ Passage par copie	807
106	Correction – ★★ Passage par adresse	808
107	Correction – ★★ Manipulation dynamique	809
108	Correction – ★★★ Définir une structure Vecteur2d	811
109	Correction – ★★★ Structure pour gérer une grille	813
110	Correction – ★★★★ Gérer un combat de personnages	816
111	Correction – ★★ Listes chaînées et ajouts	820
112	Correction – ∞ Algorithmes génétiques et problème du voyageur de commerce	822
113	Correction – ★★★★ Implémenter une liste chaînée	837

CORRECTIONS DES EXERCICES

114	Correction – ∞ Combat de créatures par fichiers	846
115	Correction – ★★ Mise en place d'un Makefile	903
116	Correction – ★★★ Profilage d'un code	906
117	Correction – ★★★ Implémentation table de hachage	907
118	Correction – ★★★ Représentation d'un lexique	916
119	Correction – ★★★ Trier une liste de points	922
120	Correction – ★★★ Benchmark de qsort	925
121	Correction – ★★★ Trier des pointeurs de fonctions	930
122	Correction – ★★★ Changer de langue	932
123	Correction – ★★★★ Simulation de combattants génériques	934
124	Correction – ★★★★ HashMap et ArrayList génériques	945
125	Correction – ★★★★ Tableaux dynamiques à n dimensions	968
126	Correction – ∞ Algorithme d'exponentiation générique	972
127	Correction – ∞ Space invaders modulaire	990
128	Correction – ★ Application opérations bit-à-bit	1020
129	Correction – ★★★ Gérer une grille sur un entier	1021
130	Correction – ★★★ Buffer pour lecture et écriture bit-à-bit	1022

CORRECTIONS DES EXERCICES

Première partie

Modalités de ce cours

Table des matières

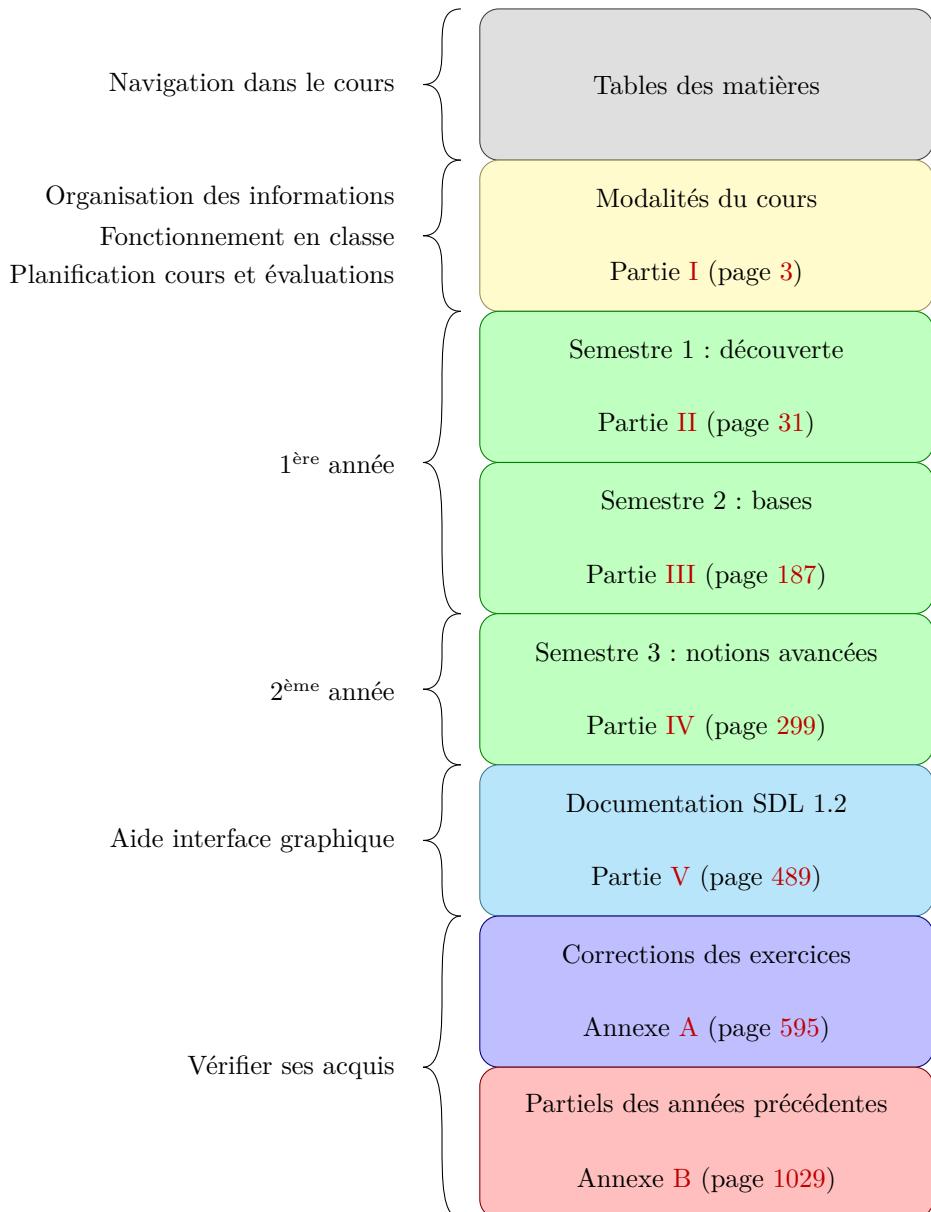
0	Préambule	5
0.1	Comment s'organise un cours ?	7
0.1.1	Séances de cours	7
0.1.2	Difficulté des exercices	8
0.1.3	Support de cours	8
0.2	Quelques règles pour travailler ensemble	9
0.2.1	Politesse	9
0.2.2	Respect	10
0.2.3	Responsabilité	11
0.3	Des conseils pour réussir l'UE Langage C ?	12
0.3.1	Première partie : découverte	12
0.3.2	Seconde partie : acquisition	12
0.3.3	Troisième partie : perfectionnement	13
0.3.4	L'assiduité	13
0.3.5	S'entraîner	13
0.4	Comment s'organisent les évaluations ?	14
0.4.1	Rendu des travaux pratiques en autonomie	15
0.4.2	Examen	18
0.4.3	Examen blanc	19
0.5	Planification cours et évaluations	19
0.5.1	Semestre 1	20
0.5.2	Semestre 2	21
0.5.3	Semestre 3	22
0.6	Projet de fin de seconde année	23
0.6.1	Étapes	23
0.6.2	Possibilités de sujet	23
0.6.3	Livrables	24
0.6.4	Évaluation	24

0 Préambule

Bienvenue au cours de langage C à l'ESGI ! Pour accompagner les cours dispensés en classe par votre professeur, vous avez le présent support de cours. Celui-ci essaiera de répondre à vos appréhensions et vous guider dans votre apprentissage du langage C cette année :

- Un sommaire cliquable en version numérique pour se rendre aux différentes sections.
- Modalités du cours partie I : règles pour un bon déroulé de séance, conseils pour appréhender son apprentissage cette année et planning des évaluations.
- Cours de la première année (Semestre 1 partie II et 2 partie III) : bases du langage.
- Cours de la seconde année (Semestre 3 partie IV) : notions plus avancées.
- En bonus : une reformulation des la documentation de SDL 1.2 et ses extensions principales (pour aller plus loin que de la programmation dans un terminal).
- Correction des exercices d'entraînement proposés dans ce support de cours.
- Sujets de l'année précédente (Semestre 1 et 2) partie VI pour se préparer à l'examen papier (pour le Semestre 3 ce sera un projet avec soutenance chapitre 0.6).

Bon courage !

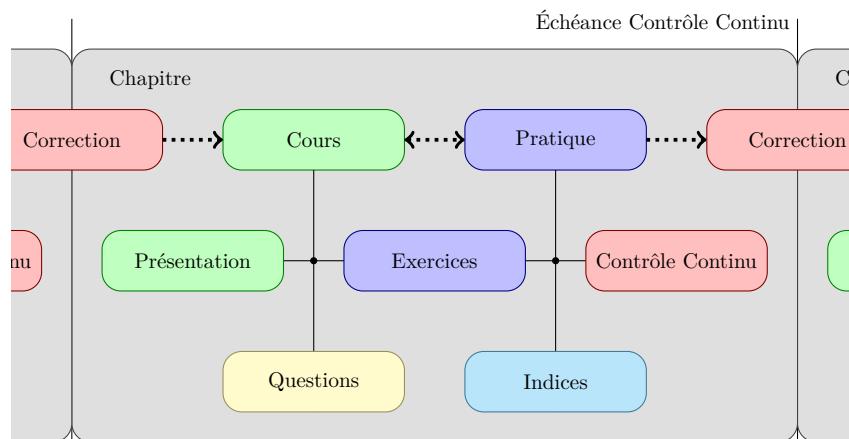


0.1 Comment s'organise un cours ?

0.1.1 Séances de cours

Les séances de cours s'articulent en général sur le schéma suivant :

1. Présentation **magistrale** d'une notion (phase de présentation / découverte avec applications pratiques intermittentes) :
 - L'enseignant présente une notion à l'aide de diapositives et reformulations au tableau selon retour des étudiants pendant la présentation.
 - Des temps d'arrêts sont dédiés à des activités d'applications rapides.
2. Temps de mise en pratique et d'**autonomie** (phase de pratique accompagnée) :
 - L'étudiant peut s'exercer sur des activités d'entraînement progressives (dont la correction est disponible en fin du support et qui peut être réexpliquée par l'enseignant si ce n'est pas fait dans la phase d'autonomie).
 - Cas particulier : dans le cas où le cours n'avance pas assez rapidement, l'étudiant a à sa disposition les chapitres suivants sur lesquels il peut s'avancer, puis un projet en Langage C par année (tout est perfectible).



0.1.2 Difficulté des exercices

La résolution des exercices ne demandent pas en général de dépasser les notions vues en classe. Si vous pensez à une solution qui semble inabordable pour la difficulté attendue, peut-être que poser à nouveau le problème donnée aiderait à une résolution plus simple.

Les niveaux de **difficulté** relatifs à la notion abordée et le gain associé pour un rendu en tant qu'assiduité :

1. ★ - trivial (0.2 point).
2. ★★ - facile (0.4 point).
3. ★★★ - moyen (0.6 point).
4. ★★★★ - difficile (0.8 point).
5. ★★★★★ - challenge (1 point).
6. ∞ - mini-projet (2 points).

0.1.3 Support de cours

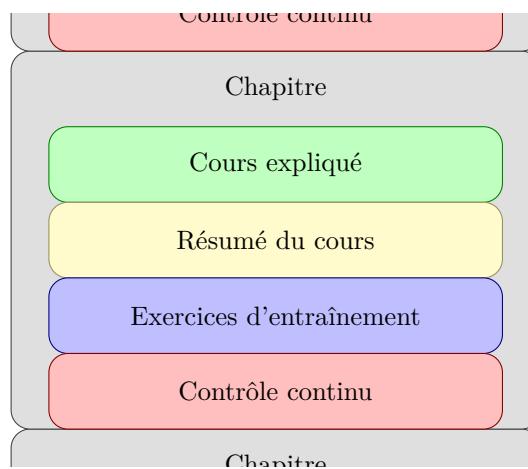
Ce présent support de cours vous accompagnera durant cette année de langage C. Ouch, mais il est énorme ce truc ! Pas de panique, on va y aller pas à pas. Il peut paraître long et bourratif au premier abord mais vous l'apprécierez probablement au fil de l'année lorsque vous aurez besoin de revenir sur une notion précédente ou vous avancer sur la suivante.

À noter que les chapitre s'étudient pas à pas. Notez que le sommaire du document est cliquable et vous permet de vous rendre directement à la section souhaitée.

Le support de cours propose en général par chapitre la structure suivante :

- Un **cours expliqué** par écrit dans le même esprit que la présentation orale en classe (utile en cas d'absence, incompréhension, pour s'avancer par exemple).
- Un **résumé** du chapitre (utile pour réviser, rechercher une information rapidement par exemple).
- Des **exercices d'entraînement** (pour pratiquer le concept abordé, tester sa compréhension).
- En fin de document, la **correction** de chaque exercice d'entraînement (pour auto-évaluer sa compréhension, potentiellement découvrir une autre manière

de résoudre le problème posé et autres).



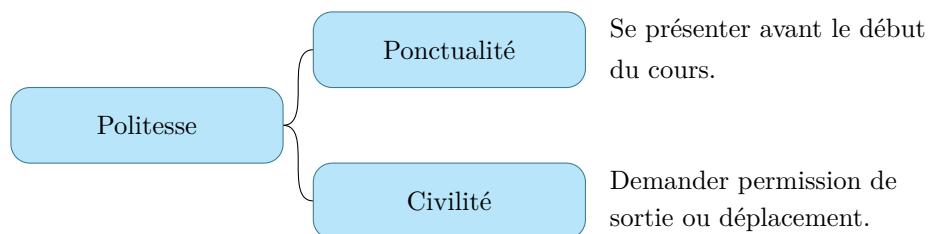
0.2 Quelques règles pour travailler ensemble

Bienvenue dans l'enseignement supérieur, votre objectif : le diplôme dans une formation que vous avez choisie.

Durant votre formation à l'ESGI, l'école souhaite à la fois vous faire accéder aux compétences techniques associées au diplôme qui vous serait décerné et vous transmettre l'attitude d'un bon professionnel lorsque vous l'y représenterez en entreprise pendant votre alternance et à l'issue de votre formation.

Ceci commence donc par quelques concepts pour permettre aux cours de se dérouler dans une ambiance optimale et propice à l'évolution de chacun.

0.2.1 Politesse

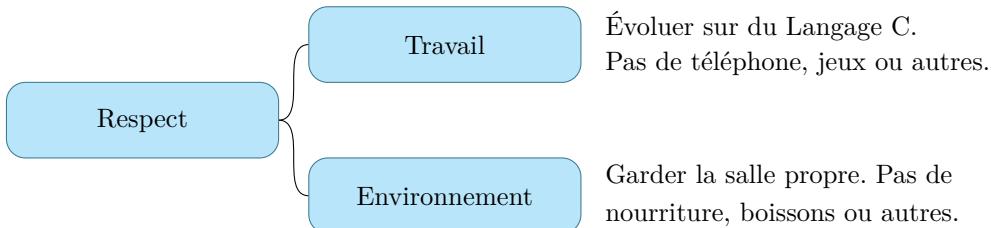


Arriveriez-vous en retard devant un film au cinéma ? À moins d'aimer le challenge de comprendre celui-ci en plein milieu et interrompre avec agacement les spectateurs déjà présents, j'en doute. Organisez vous au mieux et optez pour la manière la plus respectueuse pour rejoindre le cours. Si vous avez besoin de sortir, ne partez pas vous promener l'air de rien. Prévenez au moins votre professeur, c'est la moindre des politesses.

Extraits du règlement intérieur :

- *En cas de retards répétés, l'enseignant peut refuser l'étudiant dans son cours. L'élève entrant en retard doit être marqué « Absent ».*
- *Toute sortie de cours est définitive ; l'élève sera noté absent. Quelle que soit la raison (personnelle, appel téléphonique, ...) les élèves ne sont pas autorisés à sortir de cours.*

0.2.2 Respect



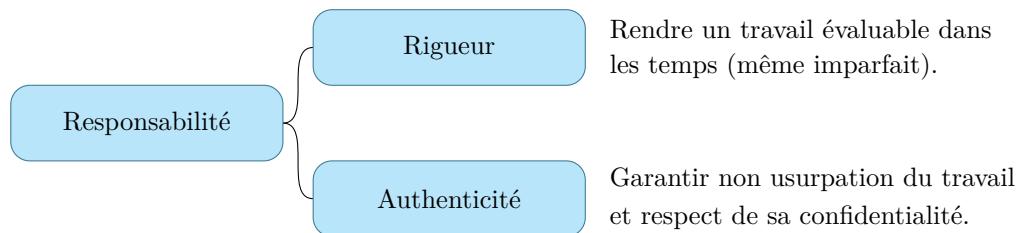
Que penseriez-vous d'être client et que le professionnel en face vous fasse attendre la fin de son combat sur Clash Royale, ne pas casser son coup Tinder, mange pendant que vous lui parlez ou s'engouffre dans une longue partie de League of Legends sur son temps de travail ? Ce serait probablement l'individu le plus professionnel que vous rencontreriez, non ? En langage C, le téléphone portable, jeux, vidéos et autres sont sources de distractions impertinentes. Notez qu'il s'agit aussi ici de respecter vos camarades, votre propre apprentissage et le formateur présent pour partager ses compétences et connaissances avec vous. Vous avez choisi l'ESGI pour progresser dans d'autres aptitudes, ayez un peu de respect et attendez la pause.

Extraits du règlement intérieur :

- *Les comportements d'indiscipline, perturbant les cours en dépit d'avertissements sont à signaler à la Direction des Études. Un intervenant est habilité à expulser un élève de cours s'il juge son comportement perturbant, dangereux ou irrespectueux.*

- *Il est rigoureusement interdit de fumer dans les locaux de l'école. Il est aussi interdit d'introduire, dans les salles de cours, boissons et nourriture ou d'utiliser un téléphone portable.*

0.2.3 Responsabilité



Surpris de recevoir la note de 0 ? Avant de jeter la faute sur une tierce personne, vérifiez que vous avez bien fait et rendu votre travail dans les modalités attendues pour sa bonne évaluation. Lorsque vous rendez un livrable à votre enseignant et à l'avenir à un client ou votre patron, faîtes attention à ce que vous rendez. Pour vous évaluer votre enseignant a besoin des sources viables de votre programme et ne pas trouver votre contenu avec pénibilité. Votre professeur a fait l'effort de vous concocter ce support de cours pour vous offrir plus de sérénité dans votre apprentissage, faîtes que ce soit réciproque.

Rendez le travail auquel vous avez contribué, le professeur se doute que vous êtes en capacité d'effectuer un copier-coller d'internet, ChatGPT, d'un camarade, de sa propre correction et de le twister si besoin. Ce n'est malheureusement pas là-dessus qu'il vous évalue et qu'il veut vous éléver par ce cours.

En effet, particulièrement en première année, votre objectif est d'acquérir les bases de la programmation, là où utiliser des méthodes détournées pour chercher à gonfler ses notes ne sera pas bénéfique pour la suite de votre apprentissage en particulier lorsque les notions se complexifient, sauf si vous recherchez à acquérir une expertise en filouterie et dans ce cas les points vous seront accordés à minima en négatif.

Essayez honnêtement, vous êtes là pour apprendre et gagner en capacités : un travail de votre part même imparfait ou incomplet sera d'autant plus apprécié et profitable pour votre enrichissement en langage C.

Extraits du règlement intérieur :

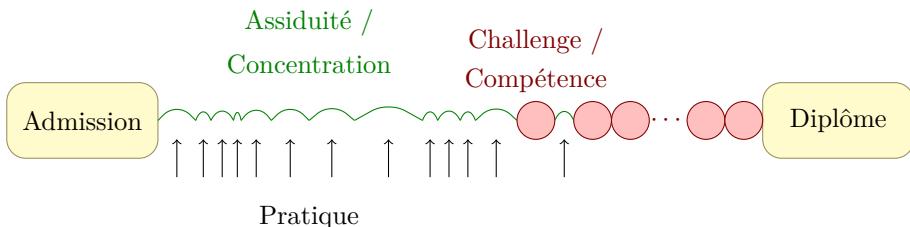
- Développer la fermeté du caractère, le sens des responsabilités, la personnalité.
- Si les enseignants constatent que des copies d'examen ou des devoirs indiquent une triche évidente, ces copies doivent être notées zéro, et un rapport remis à la Direction.

0.3 Des conseils pour réussir l'UE Langage C ?

L'apprentissage du langage C est distribué sur deux ans en trois parties. Chaque partie correspond à un **niveau d'exigence croissant** et un investissement de l'étudiant tout aussi grandissant pour réussir cette Unité d'Enseignement. Accrochez vous !

Extraits du règlement intérieur :

- Faire atteindre aux élèves un niveau de connaissances et de compétences équivalent à celui des meilleures écoles d'informatique.



0.3.1 Première partie : découverte

L'examen de la première partie du cours (partie II) aborde des notions relativement élémentaires. Celles-ci seront facilement accessibles avec une bonne assiduité en classe. L'objectif de cette première partie est d'évaluer la compréhension de la syntaxe de base du langage C et son application sur des demandes simples ne dépassant pas la notion de boucles.

0.3.2 Seconde partie : acquisition

Cependant, cette bonne assimilation des notions de la première partie ne garantissent pas un succès sans travail pour l'assimilation de la seconde partie (partie III). Un maintien du sérieux reste primordial. En effet, les notions vont avancer jusqu'aux pointeurs. À ce moment, nous commençons à effleurer les concepts du langage C. Cette partie commencera à demander une logique dans la conception du code : choix des notions à utiliser pour répondre à un problème simple.

0.3.3 Troisième partie : perfectionnement

La dernière partie (partie IV) aborde des notions plus avancées du langage C. Celle-ci a plus un but de raffinement des concepts jusque là connus. En effet, bien que les notions vues précédemment permettent de mener à bien un projet, les notions vues ici permettent de le mener avec plus de maintenabilité, d'efficacité et approfondir la connaissance des paradigmes du langage C. On attend ici d'être capable choisir ses outils dans le langage comme des briques de base pour répondre à ses besoins le plus pertinemment possible : découper son code en modules, choisir une structure de données pertinente, produire un code générique et maintenable.

0.3.4 L'assiduité

Ceci paraît trivial et pourtant ce probablement ce qui vous ferait gagner le plus de temps. Un étudiant qui écoute en classe et pose des questions permettant la précision d'une incompréhension lors du cours fera gagner à ses camarades et lui-même un temps précieux à ne pas se torturer l'esprit à lire et comprendre le cours en dehors de la séance.

Un temps est souvent donné pour essayer les exercices en classe, c'est le moment d'identifier ce qui peut poser problème et profiter de la présence du professeur pour lui demander conseil. Le temps investi en classe est du temps gagné à la maison, en particulier s'il permet de terminer les exercices avant la fin de la séance.

Extraits du règlement intérieur :

- *Les étudiants doivent être encouragés à prendre des notes succinctes lors des cours, en complément des supports de cours mis à leur disposition, et non à recopier sous la dictée le cours magistral d'un enseignant.*

0.3.5 S'entraîner

En première approche les exercices vus en classe permettent une première appréhension de la notion. Puis viennent les exercices à réaliser en autonomie pour approfondir celle-ci.

Pour se tester et s'entraîner, les sujets de l'année précédente sont disponibles en annexes VI.

Il est aussi possible de s'entraîner d'avantage sur des plateformes en ligne. Par exemple sur [CodinGame](#).

Pour réviser le cours du langage C, le plus pertinent est probablement de pratiquer. Une bonne idée peut-être de se donner un projet avec l'objectif de votre

choix qui permettrait un mise en pratique des différentes notions du cours. À noter qu'un apprentissage par cœur ne permet pas l'acquisition des automatismes et de la logique attendus lorsque vous codez. Un minimum reste tout de même requis pour connaître vos possibilités en langage C.

Extraits du règlement intérieur :

- *En dehors des cours, les élèves doivent fournir une quantité de travail conséquente, mener des travaux de recherche et d'approfondissement, compléter les connaissances acquises.*
- *Chaque semestre de cours est clos par une session de partiels. Chaque matière est soumise à un partiel final. Ce dernier est une épreuve de réflexion et d'analyse de 1h30 à 2h sur table.*

0.4 Comment s'organisent les évaluations ?

Les épreuves choisies pour le contrôle continu est donné sous forme d'un combinaison d'évaluations en classe et de devoirs maison. Votre production est à rendre sur MyGES pour la date qui y est indiquée (les rendus invalides, en retard, via un autre moyen peuvent se voir refuser). Le sujet prend la forme du rendu des exercices d'entraînement et 4 exercices :

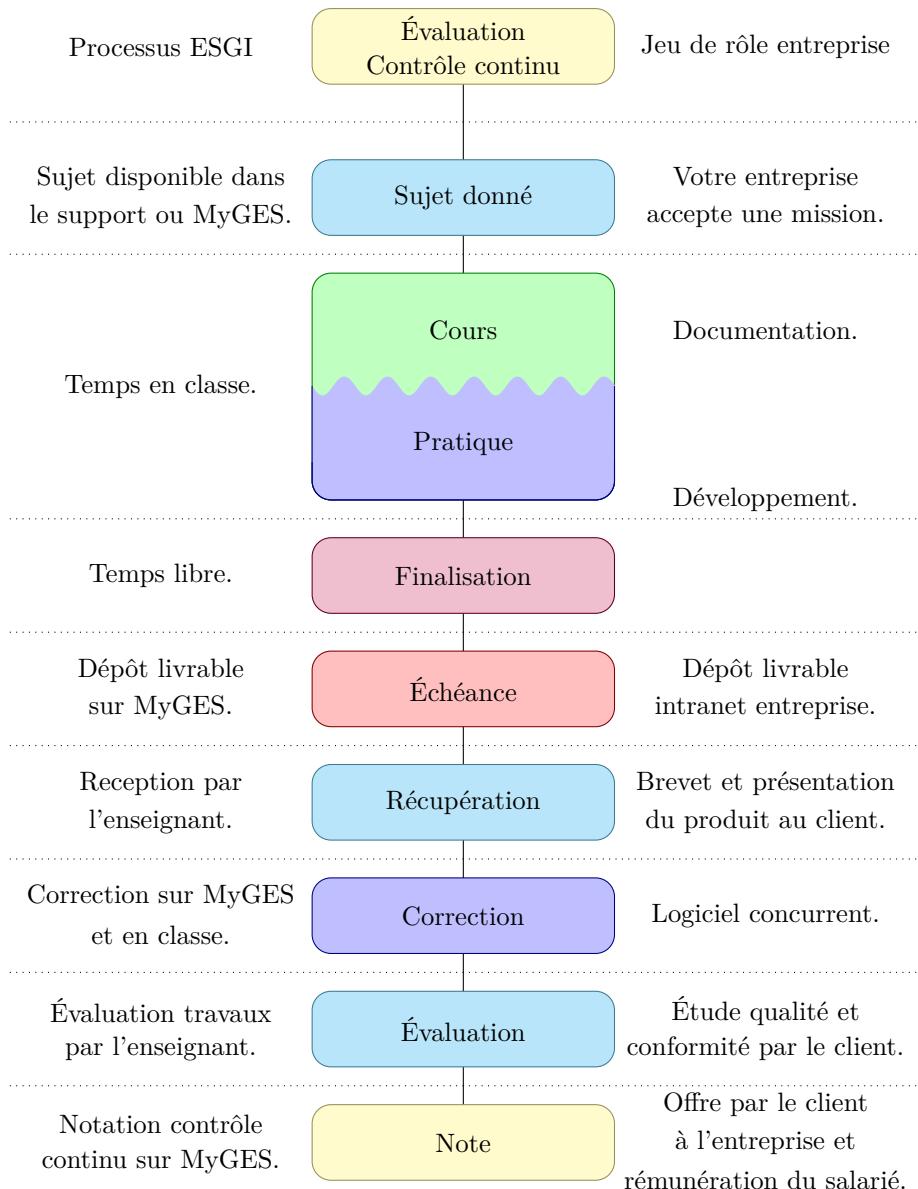
- **Évaluation en classe (1 / 2)** (5 points) : première vérification des acquis du cours (15 minutes sur papier en première année et 30 minutes sur machine en seconde année)
- **Évaluation en classe (2 / 2)** (5 points) : seconde vérification des acquis du cours (15 minutes sur papier en première année et 30 minutes sur machine en seconde année)
- **Valorisation d'assiduité** (relatif à la difficulté des 5 exercices choisis) : rendu de 5 exercices d'entraînement choisis (ce qui est valorisé est votre production personnelle : un rendu ne démontrant pas une tentative honnête comme le rendu des corrections entraînera un gain négatif à hauteur de la fraude).
- **Challenge / Recherche / Mini-projet** (5 points) : un exercice d'un niveau plus corsé, demande une investissement plus important dans la réflexion, l'expérimentation et la recherche de solution souvent via une application graphique (SDL 1.2).

0.4.1 Rendu des travaux pratiques en autonomie

Pour cette partie à continuer en devoir maison, l'étudiant peut se considérer comme en situation de développement d'un logiciel pour son entreprise. Cette entreprise ayant signé un contrat avec un client. À vous d'honorer cette mission avec professionnalisme pour votre entreprise.

Notez que ces travaux sont réalisés de votre part sous des environnements de développement très différents les uns des autres (Linux, Windows, Mac, Chrome-Book, Online GDB ou autre). Votre professeur évaluera vos travaux sous un Linux 64 bits et n'a besoin de rien d'autre que vos **sources** (fichiers .c et .h, éventuellement un exécutable pour les projets). Tout autre fichier ne sera pas considéré comme évaluables et peut être pénalisé sauf si demandé explicitement dans l'exercice.

Les exercices notés donnés sous forme de devoir maison seront à rendre sur **MyGES** dans un dépôt dédié. À noter qu'il est de la responsabilité de l'étudiant de vérifier que son travail est **évaluable** (rendu de fichiers .c cohérents avec le livrable demandé par le sujet) et rendu dans les **délais** attendus. Notez que vous pouvez rendre vos exercices tant que votre professeur ne les a pas téléchargé. Une fois notés c'est terminé, pensez donc à les rendre dans les temps sans pénaliser l'enseignant sur ses corrections. Personne ne vous courra après, vous le notifiera ou ne cherchera à récupérer votre devoir si vous le ne rendez pas sur MyGES là où attendu : vous êtes dans l'enseignement supérieur. En situation d'entreprise c'est potentiellement un client perdu qui se tournera vers la concurrence.



Règles évaluation	Engagements	Jeu de rôle entreprise
Note proportionnelle sur base de 20.	Qualité	Offre proportionnelle sur base de 20 K €.
Rendu conforme sur MyGES avant échéance.	Responsabilité	Engagement client et entreprise sur échéance.
Rendu après correction Note = 0.	Concurrentiel	Rendu retard après concurrence, client perdu.
Production extérieure ou ChatGPT, Note = 0.	Confidentialité	Diffusion externe, non breveté, client perdu.
Participants multiples, Note ajustée (Moins pénalisant si honnête).	Individualité	Bénéfice moins important pour l'entreprise, la prime est ajustée.

Le nom des auteurs doivent être indiqués en début de chaque fichier source. Ceci permet un rendu par plusieurs étudiants. Chaque participant du groupe doit rendre le devoir commun sur MyGES pour éviter les oubli de rendu d'un autre participant (si l'étudiant omet le nom des autres participants, ceci sera considéré comme malhonnête si détecté comme duplicita). À noter que si le nom n'est pas indiqué en entête de fichier, l'étudiant ne pourra pas demander par la suite d'être comptabilisé comme auteur légitime :

```
/*
 * Auteur : Kevin TRANCHO
 */
```

Le rendu par plusieurs étudiants entraîne cependant un calcul ajusté de la note par souci d'équité vis-à-vis d'étudiants rendant un travail individuel. Une

entreprise qui doit mettre plusieurs employés sur un projet réalisable par un seul perd de l'argent. Ceci est donné par la formule suivante :

$$NoteIndividuelleRendu = \max \left(NoteRendu \times \frac{5 - (NombreDeParticipants - 1)}{5}, 0 \right)$$

Nombre de participants	Note maximale
1	5
2	4
3	3
4	2
5	1
6+	0

Dans le cas où une même composition (ou inspiration bien trop suspecte pour être honnête) se retrouverait de manière inadéquate (non déclarée et détectée par l'enseignant) sur plusieurs rendus d'individus différents, le nombre de duplcitas comptera comme de participants auquel 1 sera ajouté pour tentative de rendu peu honnête.

À noter que ChatGPT ne respecte pas d'accord de confidentialité et partage ses solutions à un nombre élevé d'étudiants tout comme un étudiant qui partagerait son code sur Discord, ceci entraîne un groupe de rendu très important et un ajustement en général à 0. À vous de savoir doser l'aide et l'utilisation d'outils sans mettre en porte-à-faux votre entreprise vis-à-vis d'un client et de la brevetabilité d'un produit.

0.4.2 Examen

Première année

Les deux semestres de la première année se clôturent par un examen sur **papier**. Pour celui-ci toute assistance que vous pourriez avoir en classe et chez vous n'existe plus. C'est vous contre votre feuille. Vous pouvez regarder ou vous tester sur les sujets tombés précédemment en partie VI. À noter que la séance avant l'examen, un sujet blanc sera travaillé ensemble comme avant-goût de la situation réelle.

Seconde année

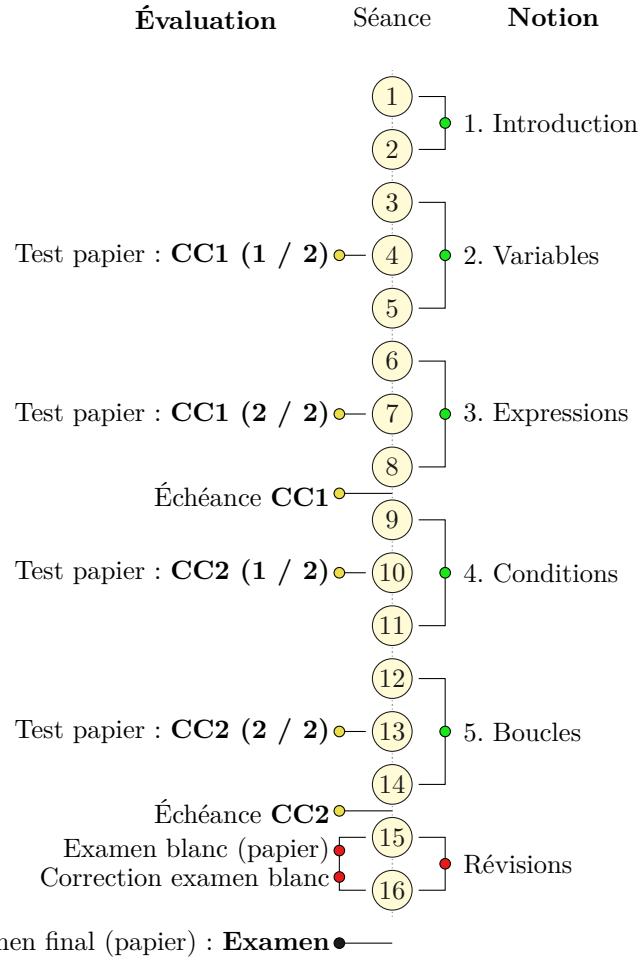
En seconde année, vous êtes évalué sur un projet en groupe dont vous pourrez choisir le sujet (sauf si vous y tardez votre enseignant l'imposera). Vous devrez présenter votre travail lors d'une soutenance.

0.4.3 Examen blanc

La **séance de fin de Semestre (1 et 2)**, nous organiserons une évaluation papier d'1h30 surveillée par le professeur pendant la première partie de la séance. Puis le temps d'1h30 après la pause sera dédié à la correction magistrale de cette évaluation avec correction de votre copie par un camarade en vue de préparer l'examen final.

0.5 Planification cours et évaluations

Vous trouverez ici la planification du cours en amont : notions abordées et moment des évaluations en fonction de la séance traitée. Chaque numéro correspond à une séance de 1h30. Une planification plus explicite relative à votre classe vous sera donnée. Attention, l'emploi du temps relatif au cours explicite peut varier.

0.5.1 Semestre 1**Partie II**

0.5.2 Semestre 2

Partie III

The diagram illustrates a vertical flow of tasks:

- At the top is a horizontal row of three labels: "Évaluation", "Séance", and "Notion".
- Below this, a series of numbered circles (1 through 10) are arranged vertically, connected by lines that branch out to the right.
- Circle 1 is associated with "6. Fonctions".
- Circles 2 and 3 are associated with "7. Tableaux".
- Circles 4 and 5 are associated with "8. Pointeurs".
- Circle 6 is associated with "Échéance CC1".
- Circle 7 is associated with "Examen blanc (papier)".
- Circle 8 is associated with "Correction examen blanc".
- Circle 9 is associated with "Révisions".
- Circle 10 is associated with "Examen final (papier)".

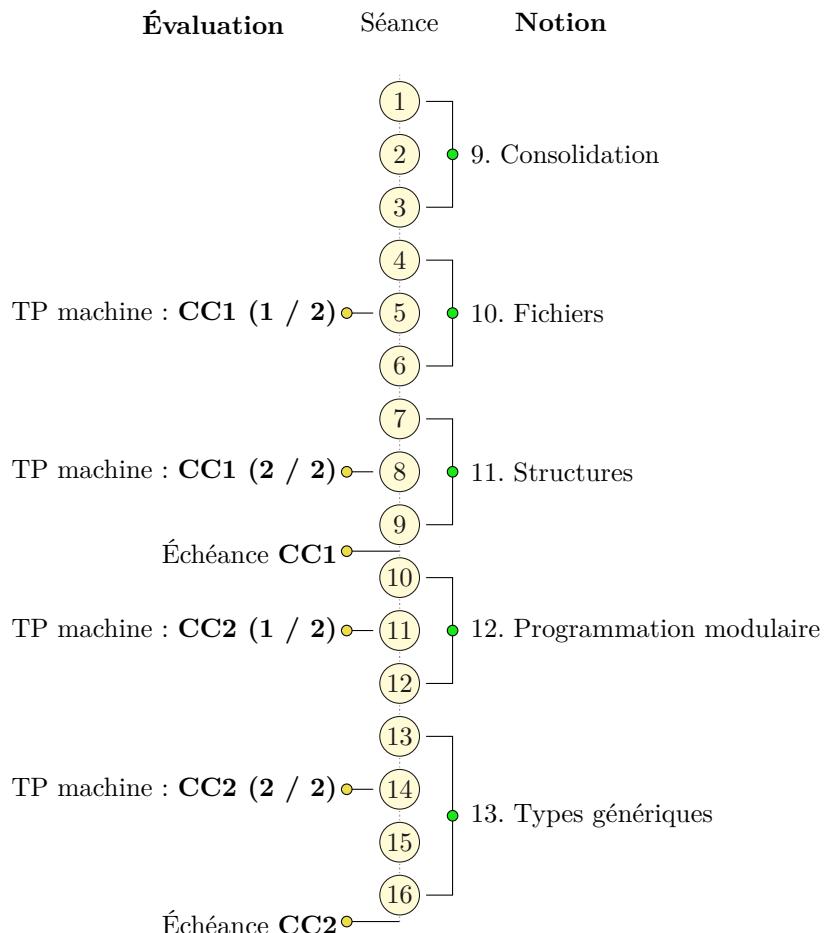
On the left side of the diagram, there are two "Test papier" sections:

- "Test papier : CC1 (1 / 2)" is positioned between circles 1 and 4.
- "Test papier : CC1 (2 / 2)" is positioned between circles 7 and 8.

On the far left, the text "Examen final (papier) : Examen" is followed by a large black circle, which is connected by a line to the bottom of circle 10.

0.5.3 Semestre 3

Partie IV



Rendu et soutenance projet final : **Examen** •

0.6 Projet de fin de seconde année

0.6.1 Étapes

Nous allons procéder en quelques étapes pour ce projet :

1. **(Échéance : 21 Octobre 2024)** S'armer d'un ou deux camarades pour attaquer le projet.
(Dans le cas où vous n'êtes pas affecté à un groupe de 2 à 3 personnes dans les temps sur MyGES, le professeur procédera à un matchmaking automatique).
2. **(Échéance : 03 Novembre 2024)** Choisir une thématique et un sujet puis les faire valider par le professeur.
(Dans le cas où ceux-cis n'ont pas été donnés ou validés par le professeur dans les temps, le sujet sera imposé).
3. **(Échéance : 24 Novembre 2024)** Rendu d'une roadmap du projet : organiser le projet (répartition des tâches dans le groupe), listing des fonctionnalités, planification dans le temps.
4. **(Échéance : 05 Janvier 2025)** Rendu d'un prototype du projet (preuve de concept, base à peaufiner).
5. **(Échéance : 05 Février 2025)** Rendu final du projet.
6. **(Mois de Février)** Soutenance.

0.6.2 Possibilités de sujet

Vous pouvez choisir un sujet qui met en avant des concepts algorithmiques ou relatifs à des structures de données. Ceci peut être du benchmarking, une utilisation et mise en évidence de celle-ci dans un contexte donné. Ceci peut s'intéresser à des structures chaînées (listes, arbres, graphes) et mettre en évidence leur manipulation dans un cas pratique ou comparer à d'autres structures sur des tailles de données pertinentes.

Une possibilité est de réaliser un jeu vidéo à l'aide d'une bibliothèque graphique telle que SDL. À vous de proposer des mécaniques intéressantes qui permettent l'utilisation de type structurés et de types plus génériques. Ceci peut aussi permettre de s'intéresser à des méthodes algorithmiques et structures de données

pour gérer un nombre d'entités important dans le contexte que vous proposerez.

Vous pouvez aussi vous intéresser à la programmation système. Ceci peut décliner sur la reproduction de commandes Linux, la simulation d'un système de fichiers.

Il est aussi possible de s'intéresser à des fonctionnalités plus orientées web et base et données dont la récupération de pages web avec cURL ou des appels à une base de données via MySQL.

0.6.3 Livrables

Vous devrez rendre sur MyGES les documents suivants :

1. (**Échéance : 03 Novembre 2024**) Un document rapide donnant le sujet de votre projet (1 page) après validation du sujet par le professeur.
2. (**Échéance : 24 Novembre 2024**) Fichier roadmap du projet : organiser le projet (répartition des tâches dans le groupe), listing des fonctionnalités, planification dans le temps.
3. (**Échéance : 05 Janvier 2025**) Sources fonctionnelles et viables en langage C du **prototype** avec Readme et Makefile.
4. (**Échéance : 05 Février 2025**) Sources fonctionnelles et viables en langage C du **projet finalisé** avec Readme / rapport, Makefile et accès à un git.

0.6.4 Évaluation

Votre score final pourra varier entre 0 et 21 sur 20 points. L'obtention d'une note supérieure à 20 ne pourra pas être enregistrée sur MyGES mais vous donnera toute justification pour demander à votre professeur de mettre votre projet à l'honneur. À noter que votre score sera individuel : en effet vous devrez justifier dans le rapport et à l'aide d'un git de l'investissement de chacun pour une attribution correcte des points. En effet, 10 points sont communs au groupe (sous réserve d'avoir démontré une activité dans celui-ci) et 11 points sont individuels.

À noter que si l'est détecté des incohérences dans le projet (fonctionnalités non réalisées par le groupe, inégalité du travail, recyclage d'un code non réalisé pour le projet ou prise de liberté sur le sujet annoncé / groupes définis ou autres) la notation pourra se voir lourdement sanctionnée ou ajustée pour non respect des consignes, travail inéquitable ou tentative malhonnête.

(... / 9 points) Code**(... / 7 points de groupe)****(... / 2 points individuels)**

Cette année et l'année précédente nous avons étudié des concepts qui vous permettent l'écriture d'un code en langage C. La qualité du code et le niveau des concepts utilisés comme démonstration des capacités techniques du candidats seront valorisés. Veillez à garder tout aussi lisible que pertinent.

Dans le code nous attendons :

- **(... / 1 point : individuel)** Un code propre, indenté, sans warnings et qui compile (avec gcc sous une machine Ubuntu 64 bits).
- **(... / 1 point : individuel)** De la documentation : un autre programmeur ou vous-même doivent pouvoir continuer votre projet après son rendu.
- **(... / 1 point)** Une découpe en modules pertinente.
- **(... / 1 point)** Structuration du code lorsque pertinent : types structurés, énumérations.
- **(... / 1 point)** Généricité du code lorsque pertinent : pointeurs de fonctions, types génériques.
- **(... / 4 points)** Relatif à la thématique :
 - **(... / 2 points)** Utilisation et mise en place de bibliothèques (graphique, système, base de données et autres).
 - **(... / 2 points)** Qualité de l'implémentation (structures de données, généricité, benchmarking, niveau conceptuel).

(... / 7 points) Fonctionnalités**(... / 1 point de groupe)****(... / 6 points individuels)**

Votre pratique du langage C et de l'algorithme vous a permis de développer une logique et capacité réussir avec succès le codage de fonctionnalités souhaitées. Vous démontrez ici votre capacité à faire vivre votre pensée dans une programme automatique et efficace.

Côté fonctionnalités, nous attendons :

- **(... / 1 point)** Un programme efficace (pas de latence inexplicable) et qui ne crash pas (ou du moins précise la raison de l'arrêt du programme pour réussir à le faire fonctionner).

- (... / **2 points** : individuel) Appréciation de l'atteinte des objectifs fixés dans la roadmap.
 - (... / **2 points** : individuel) Appréciation de la pertinence du prototype (qualité / niveau technique de la réalisation).
 - (... / **2 points** : individuel) Appréciation de la pertinence du rendu final (qualité / niveau technique de la réalisation).

(... / 5 points) Soutenance et rapport

La soutenance et rapport ont pour but d'aider à la prise en main de votre projet et comprendre votre cheminement dans sa conception. Le rapport peut être matérialisé par un fichier **README.md** et / ou un fichier **.pdf**. Montrez que votre projet a été réalisé en équipe avec professionnalisme.

Nous attendons du rapport et de la soutenance qu'ils nous informent sur :

- (... / 0.5 point) Comment lancer votre projet ? Quelles sont ses dépenses ?
 - (... / 1 point) Le contexte et l'organisation de votre groupe pour répondre à la problématique : avez-vous établi un planning ? Qui a fait quoi et pourquoi ?
 - (... / 1 point : individuel) Pourquoi avez-vous organisé le code de cette manière ? Quels sont les éléments du cours ou du langage C que vous avez utilisé pour le mener à bien ? Ces choix ont-ils été pertinent pour avancer plus rapidement dans votre code, garder en maintenabilité ?
 - (... / 1 point : individuel) Quelles fonctionnalités proposez-vous dans ce projet ? Comment les avez-vous optimisées ? Valorisez l'apport de vos connaissances en algorithmique et leur intérêt pour ce projet en langage C.
 - (... / 0.5 point) Avez-vous rencontré des difficultés techniques, organisationnelles, relationnelles pour réaliser ce projet ? Comment les avez-vous dépassées ?
 - (... / 1 point : individuel) La concordance du projet avec votre formation. Quelles sont les aptitudes qui ont été requises de vous et comment votre formation à l'ESGI (ce cours ou d'autres) vous a aidé ou fait défaut ? Avez-vous aimé réaliser ce projet ? Pourquoi ?

Vous êtes libre de la présentation et démonstration de votre projet le jour de la soutenance. Vous aurez 10 minutes de présentation, suivies de 5 minutes de potentielles questions par le Jury. La soutenance a pour but de défendre / vendre

otre projet au Jury en vue qu'il investisse des points dans celui-ci. En plus de votre projet, votre objectif est de démontrer les compétences techniques liées et acquises Langage C de chaque participant du groupe.

Dans le cas où vous manqueriez d'inspiration pour votre présentation, celle-ci peut prendre les formes suivantes :

- Présentation rapide via des diapositives (fonctionnalités, répartitions des tâches, éléments techniques pertinents), suivie d'une démonstration technique rapide : preuve de concept (mise en avant des éléments abordés ou complémentaires).
- Démonstration technique avec mise en avant des fonctionnalités proposées (auteur, optimisations de la méthode, choix de conception et d'implémentation).

Sur la phase de questions, vous pouvez être interrogés sur le fonctionnement d'un morceau de code, le code correspondant à une fonctionnalité, justifier une implémentation, justifier une fonctionnalité, justifier l'organisation de votre groupe et la répartition des tâches, poussé dans une réflexion sur votre implémentation et ses limitations, expliquer la démarche derrière une fonctionnalité ou autre question / vérification pouvant permettre à la bonne évaluation de votre projet.

Bon courage !

Deuxième partie

Découverte

Table des matières

1	Introduction	35
1.1	Informatique	35
1.2	Algorithmique	36
1.2.1	Activité introductory à la programmation	36
1.3	Langage C	37
1.3.1	Motivation	37
1.3.2	Compilation	38
1.3.3	Édition des liens	39
1.4	Première application console	39
1.4.1	Code source .c	39
1.5	Préparer ses outils	41
1.5.1	Installation gcc	42
1.5.2	Compilation avec gcc	47
1.5.3	Utiliser gcc sous Windows	48
1.5.4	Mac alternative : Xcode	51
1.5.5	Online GDB : compilateur en ligne	52
1.6	Lire la documentation	52
1.6.1	Dans son terminal	52
1.6.2	En ligne sur cppreference.com	53
1.7	Résumé	54
1.8	Entraînement	56
2	Variables	57
2.1	Introduction et motivation	57
2.2	Variables typées	59
2.2.1	Déclaration	59
2.2.2	Entiers signés	60

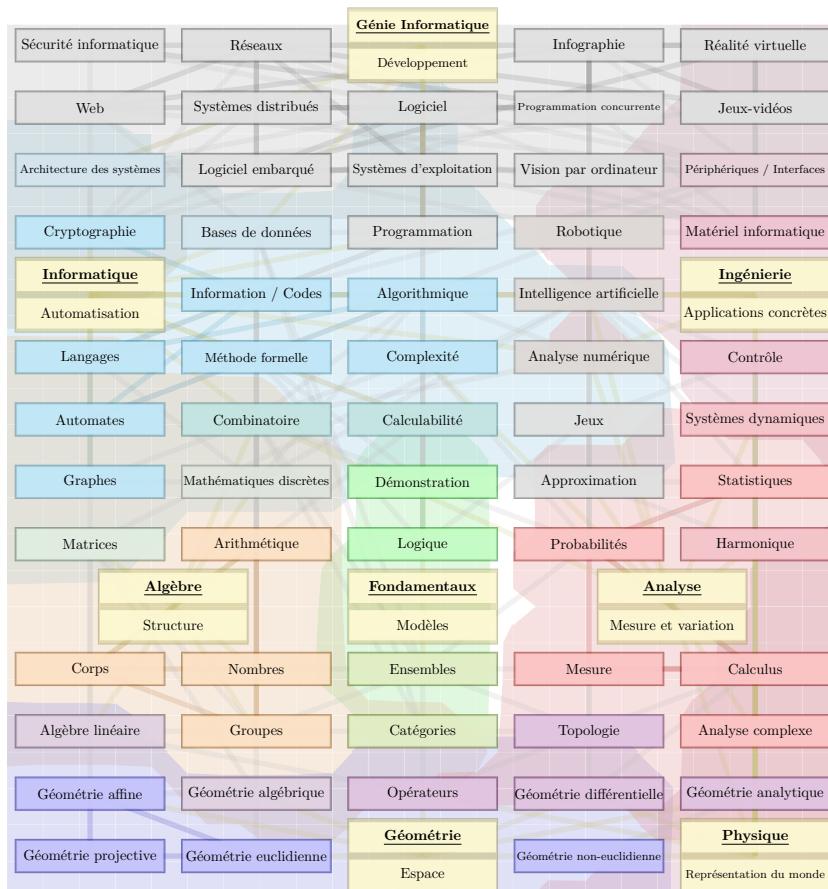
2.2.3	Entiers non-signés	61
2.2.4	Nombres à virgule flottante	61
2.2.5	Constantes	62
2.3	Format d'affichage	63
2.3.1	Printf	63
2.3.2	Formats pour entiers	63
2.3.3	Formats pour flottants	67
2.4	Adresse d'une variable	69
2.5	Scanf	70
2.6	Résumé	73
2.7	Entraînement	76
3	Expressions	91
3.1	Opérations classiques	91
3.1.1	Addition	91
3.1.2	Soustraction et multiplication	93
3.1.3	Compatibilité entre entiers et flottants	93
3.2	Division	94
3.2.1	Division entière	94
3.2.2	Modulo : reste de la division euclidienne	94
3.2.3	Division fractionnaire	95
3.3	Coercition : un changement de type	96
3.4	Réaffectation d'une variable avec opérateur	99
3.5	Résumé	102
3.6	Entraînement	104
4	Conditions	117
4.1	Sélection d'instructions avec if	117
4.2	Comparaisons	120
4.3	Opérateurs booléens	122
4.3.1	Intersection	122
4.3.2	Union	124
4.3.3	Négation	125
4.4	Tertiaire : expression sous condition	127
4.5	Switch : se brancher un à résultat	128
4.6	Résumé	132
4.7	Entraînement	133

5 Boucles	157
5.1 While : répétition sous condition	157
5.2 Do while : répétition si besoin	158
5.3 For : un pas après l'autre	159
5.4 Contrôle de la boucle : break or continue	161
5.5 Résumé	164
5.6 Entraînement	165

1 Introduction

1.1 Informatique

Aujourd’hui entourés d’outils numériques en tout genre : téléphone portable, ordinateur et autres. Leur utilisation est devenue si triviale. Mais vous voilà à l’École Supérieur de Génie Informatique. C’est que vous avez choisi d’en prendre le contrôle ! Vous avez probablement entendu parler de codage ou encore d’algorithmes ? Que cachent ces mots ?



L'informatique à l'origine une branche des mathématiques dont l'étude porte sur l'automatisation qui s'est à finalement trouvé ses applications et son indépendance via son implémentation dans des machines physiques concrètes. L'informatique est un monde bien plus vaste qu'il n'y paraît, mais pas de panique : l'ESGI vous a concocté un programme pour vous guider dans la découverte de cette science, en acquérir les bases puis choisir les thématiques vers lesquelles vous souhaiteriez vous spécialiser ! Le voyage est difficile, mais en vaut la peine, accrochez-vous !

1.2 Algorithmique

Lorsque nous parlons d'informatique, la première idée est "ordinateur". Or, il se trouve que l'ordinateur n'est qu'un outil permettant l'implémentation d'une logique.

1.2.1 Activité introductive à la programmation

Si nous vous demandions de trier une liste de nombres du plus petit au plus grand, vous y arriveriez aisément, non ? Est-ce aussi le cas si vous deviez l'expliquer à un camarade ou pire encore le formaliser pour une machine ? Testons l'exercice avec le français, votre langage de programmation :

Exercice 1 (Formaliser une logique).

1. Découper une feuille de papier en au moins 8 morceaux et y écrire des nombres.
2. Mélanger ces nombres.
3. Trier ces nombres.
4. Expliquer oralement à un camarade (voisin) la méthode choisie pour trier ces nombres et lui demander de la mettre en œuvre.
5. Écrire sur papier les instructions à suivre pour que tout camarade puisse trier une liste de nombres selon la méthode choisie.
6. Faire tester la méthode à votre professeur qui ne connaît que le jeu d'instructions suivant : déplacer des papiers avec ses mains, comparer des papiers deux à deux.

Cet exercice a conduit à l'écriture d'un **algorithme** en langage compréhensible par un humain. En effet, c'est comme une recette de cuisine : à partir d'éléments donnés, il suffit de suivre une suite d'**instructions** pour arriver automatiquement

à un résultat souhaité.

Des exemples d'algorithmes ont déjà été rencontré durant la scolarité comme l'addition, la soustraction, la multiplication et la division posées. En effet, ces méthodes prennent au moins deux nombres et donnent automatiquement le résultat attendu. Il en est de même pour le plus explicite algorithme d'Euclide qui calcule le PGCD de deux nombres à l'aide de divisions successives.

Effectuer ces calculs est souvent humainement fastidieux d'autant plus pour des grands nombres. Ceci motive l'usage de la calculatrice par exemple. Mais pour ceci il faut avoir été capable d'indiquer à une machine comment réaliser ces tâches de manière automatique. Cette communication avec la machine peut se faire à l'aide d'un **langage de programmation**. Pour notre cours, nous avons choisi le langage C.

1.3 Langage C

1.3.1 Motivation

De nombreux langages de programmation existent que ce soit pour permettre de jouer aux derniers jeux vidéos, aller sur le réseau social à la mode, sécuriser nos transactions bancaires et tant d'autres choses.

Il existe une multitude de langages de programmation qui répondent à plusieurs besoins, certains vont préférer une écriture de code rapide au détriment du contrôle sur la machine et potentiellement de l'efficacité, d'autres vont se concentrer sur les performances que l'on peut obtenir mais ceci demandera souvent de garder un langage qui offre beaucoup de contrôle sur la machine et demande un plus long temps d'écriture.

L'initiation commencera avec le langage C, pourquoi ce choix ?

Nous avons décidé que vous mettriez les "mains dans le cambouis" ! En effet, en langage C vous allez pouvoir opérer sur la mémoire en manipulant des adresses et ainsi avoir une idée de comment on peut la gérer. Ceci est en général fait de manière cachée dans d'autres langages de programmation.

Le langage C, c'est un assemblage de concepts de bases logiques et relativement proches des actions d'un ordinateur moderne pour commencer à appréhender son fonctionnement. De plus le langage C reste un langage qui offre de bonnes performances du fait d'être compilé en langage machine (ordres donnés au processeur

par la lecture d'une bande de 0 et de 1). Ceci vous fait ensuite une application directement lisible de la machine, ce qui offre des performances qui dépendent de vous et non d'un interpréteur.

Le langage C est à la base d'une grande majorité de langages de programmation impérative (instructions exécutées successivement) aujourd'hui. Suite à votre apprentissage du langage C et ses paradigmes vous pourrez prendre du recul sur celui-ci pour mieux apprécier d'autres langages de programmation.

Historiquement, le langage C a été inventé par Brian Kernighan, Dennis Ritchie et Ken Thompson dans l'idée de réécrire le système d'exploitation UNIX en 1970 après une version en assembleur en 1969. Aujourd'hui, ce langage est encore utilisé pour des applications demandant de hautes performances comme les systèmes d'exploitation classiques et embarqués (Linux depuis 1991), les jeux vidéos et moteurs de rendu 3D (comme Blender) par exemple.

Nous noterons quelques versions importantes :

- 1970 : Version pour écrire UNIX (Dennis Ritchie et Kenneth Thompson).
- 1972 : Version distribuée par les laboratoires Bell (Brian Kernighan, Dennis Ritchie et Kenneth Thompson).
- 1989 : Norme ANSI (C89), normalisation pour compatibilité sur toutes les machines.
- 1999 : Norme C99, tableaux de taille variable, commentaires à la C++.
- 2011 : Norme C11, ajout de fonctionnalités pour la programmation multi-thread.

Dans le cadre de notre cours, nous resterons sur la norme **ANSI**, suffisante pour les paradigmes étudiés, bien que vous soyez invités à explorer davantage les fonctionnalités du langage.

1.3.2 Compilation

Le langage C reste un langage qui reste entre la compréhension humaine et la possibilité d'interprétation facile par la machine mais ne peut pas être exécuté directement. Pour que la machine puisse effectuer les actions souhaitées, il est nécessaire de transformer ce code en un ensemble de directives que la machine peut exécuter, plus concrètement en langage machine (assimilable au langage assembleur). On dira que ce code directement lisible par la machine est **compilé**.

1.3.3 Édition des liens

La dernière étape pour obtenir un **exécutable** fonctionnel est l'édition des liens / **linkage**. Le linkage c'est le référencement vers des fonctionnalités externes au code écrit comme par exemple la bibliothèque standard `stdio.h`. C'est comme avoir à disposition des outils accessibles potentiellement fabriqués par quelqu'un d'autre et des recettes de cuisines ou parties de recettes à disposition. Ceci est une étape d'édition de liens qui indique à l'avance au programme où il faudra aller pour les utiliser.

1.4 Première application console

1.4.1 Code source .c

Pour indiquer à la machine le comportement à adopter, nous devons écrire du **code source** en langage C. C'est dans un fichier `.c` que nous écrivons les différentes **instructions** en langage C comme illustré dans `hello_esgi.c` ci-dessous. Un tel fichier doit être ouvert et édité avec un environnement de développement C (Integrated development environment : IDE conseillé sous Windows) ou avec le bloc note par défaut et non un logiciel de traitement de texte. Il en existe un grand nombre et le choix s'orientera vers celui qui vous convient le mieux : [Atom](#), [CLion](#), [CodeBlocks](#), [Emacs](#), [Notepad++](#), [Qt Creator](#), [Sublime Text](#), [Vim](#), [Visual Studio Code](#) ou autre.

```
00_introduction/hello_esgi.c
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     printf("Hello ESGI !\n");
6     exit(EXIT_SUCCESS);
7 }
```

Ici, les deux premières lignes indiquent les bibliothèques où trouver les procédures pour l'affichage et la sortie du programme. C'est comme indiquer à l'avance que l'on va avoir besoin d'ouvrages pour utiliser des recettes de cuisine sans avoir à en écrire leur contenu car ces recettes seront attachées au programme au linkage.

```
#include <stdio.h>
```

La bibliothèque `stdio.h` correspond à la bibliothèque **standard** d'entrées et sorties (*input* and *output*), elle fournit notamment la procédure de `printf` (ligne 5).

```
#include <stdlib.h>
```

La bibliothèque `stdlib.h` correspond à la bibliothèque **standard** générale, elle fournit notamment la procédure de `exit` (ligne 6).

```
int main() {  
    ...  
}
```

La procédure `main` est l'entrée du programme : lorsque l'application est lancée, on exécute les instructions qui remplacent les ... donnés entre les accolades qui suivent `main`.

Ceci fait que lorsque l'application se lance, on exécute la ligne suivante :

```
printf("Hello ESGI !\n");
```

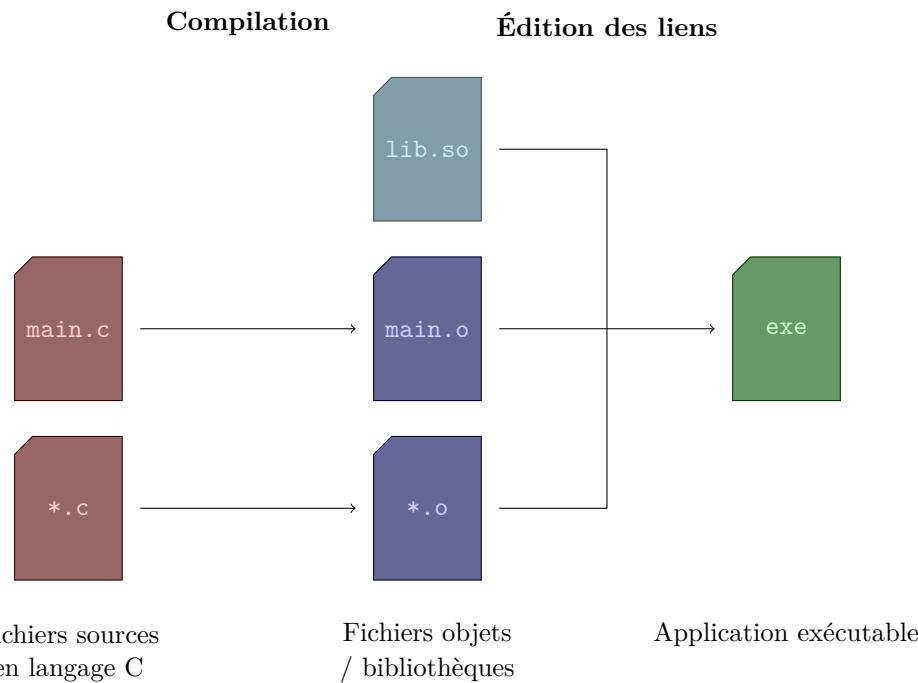
ce qui imprimera à l'écran "Hello ESGI!" suivi d'un retour à la ligne. Puis la ligne :

```
exit(EXIT_SUCCESS);
```

qui termine le programme indiquant que c'est un arrêt normal de celui-ci.

Notons qu'en langage C, lorsqu'on a terminé d'écrire notre instruction, on l'indique à l'aide d'un point virgule ;. Ceci sera probablement un oubli fréquent au début de votre formation et vous vaudra d'en être informé par le compilateur par `error: expected ';'.`

Cependant ce code en tant que tel ne peut pas être lu directement par la machine. Il nécessite d'être transformé en langage machine à l'aide d'un **compilateur** et d'un **linker**. Pas de panique, ceci se fait généralement sans s'en soucier (une commande ou une pression de bouton) à l'aide des outils dédiés.



1.5 Préparer ses outils

Bon, c'est bien beau cette théorie, mais ce serait plus pratique que la machine lance le programme histoire de la voir, non ?

Pour ceci vous devrez installer les outils énoncés plus haut : compilateur et éditeur des liens. Notez qu'ils sont en général regroupés comme une même application. Pour suivre au mieux le cours, l'installation de **gcc** vous est recommandée. En cas de difficultés, pas de panique, **Online GDB** est un compilateur en ligne qui fera l'affaire.

Notez que vous pourriez apprécier l'auto-complétion de certains outils. Cependant si vous débutez et vu que vous serez évalué sur papier la première année, je vous conseille d'avoir un éditeur avec seulement l'indentation automatique pour vous forcer à prendre en main la syntaxe et ce qui est nécessaire pour faire du code C compilable.

Exercice 2 (★ Compiler Hello ESGI!).

La suite de ce chapitre propose différentes solutions pour installer un compilateur sur votre ordinateur en fonction de votre système d'exploitation.

Choisissez la solution proposée ou extérieure qui vous convient le mieux pour vous permettre de suivre ce cours sans encombre.

En cas de problème bloquant sans solution, notez que vous avez la possibilité d'utiliser un compilateur en ligne : [Online GDB](#) (ou une machine virtuelle Linux : comme une Debian ou Ubuntu).

1.5.1 Installation gcc

Linux

Sous Linux, nous proposons d'installer `gcc` pour compiler et linker un programme écrit en C.

Ceci à par la saisie des instructions suivantes dans un terminal :

```
sudo apt update  
sudo apt install build-essential  
sudo apt-get install manpages-dev
```

Il est ensuite possible de vérifier que l'installation est réussie en lançant `gcc` :

```
gcc --version
```

ce qui doit renvoyer un retour similaire au suivant :

```
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0  
Copyright (C) 2017 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions.  
  ↳ There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR  
  ↳ PURPOSE.
```

Windows : MSYS2

Pour vous permettre de suivre au mieux le cours de la même manière que si vous utilisez un Linux, je vous conseille d'installer vos outils à l'aide de **MSYS2**. Ici, vous avez à suivre les instructions d'installation puis de lancer MSYS2. Ensuite, vous pourrez saisir les commandes suivantes pour installer vos outils :

```
pacman -Syy  
pacman -S --needed base-devel mingw-w64-x86_64-toolchain  
pacman -S --needed base-devel mingw-w64-x86_64-toolchain  
pacman -S base-devel gcc make man-pages-posix cmake
```

Il est ensuite possible de vérifier que l'installation est réussie en lançant `gcc` dans MSYS2 :

```
gcc --version
```

ce qui doit renvoyer un retour similaire au suivant :

```
gcc.exe (Rev3, Built by MSYS2 project) 12.1.0
```

Vous avez maintenant accès à un environnement viable pour travailler en langage C comme sous Linux.

Windows : MSYS2 pas-à-pas

La procédure pouvant être fastidieuse et semée d'embûches pour un total débutant, nous détaillons ici les étapes à suivre pour une bonne installation :

1. Télécharger MSYS2 depuis sa page de téléchargement au lien suivant :
<https://www.msys2.org/>

MSYS2

Software Distribution and Building Platform for Windows

MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software.

It consists of a command line terminal called [mintty](#), bash, version control systems like git and subversion, tools like tar and awk and even build systems like autotools, all based on a modified version of [Cygwin](#). Despite some of these central parts being based on Cygwin, the main focus of MSYS2 is to provide a build environment for native Windows software and the Cygwin-using parts are kept at a minimum. MSYS2 provides up-to-date native builds for GCC, mingw-w64, CPython, CMake, Meson, OpenSSL, FFmpeg, Rust, Ruby, just to name a few.

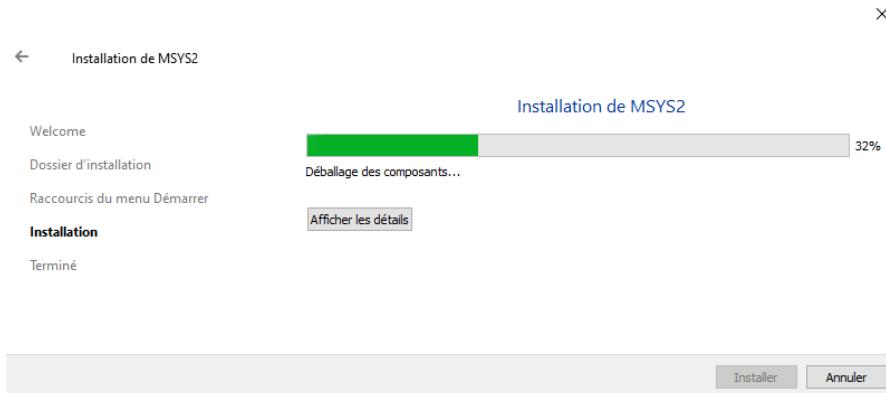
To provide easy installation of packages and a way to keep them updated it features a package management system called [Pacman](#), which should be familiar to Arch Linux users. It brings many powerful features such as dependency resolution and simple complete system upgrades, as well as straight-forward and reproducible package building. Our package repository contains [more than 2900 pre-built packages](#) ready to install.

For more details see '[What is MSYS2?](#)' which also compares MSYS2 to other software distributions and development environments like [Cygwin](#), [WSL](#), [Chocolatey](#), [Scoop](#), ... and '[Who Is Using MSYS2?](#)' to see which projects are using MSYS2 and what for.

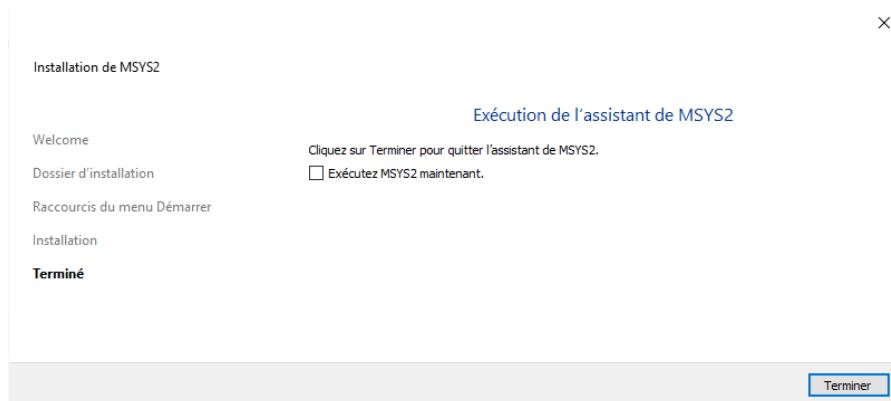
Installation

1. Download the installer: [msys2-x86_64-20230718.exe](#)

2. Installer MSYS2 :



3. Fermer l'installation de MSYS2 (ne pas cocher l'option d'ouvrir MSYS2 : ceci n'ouvrira pas le bon terminal) :



4. Lancer le terminal général de MSYS2 :

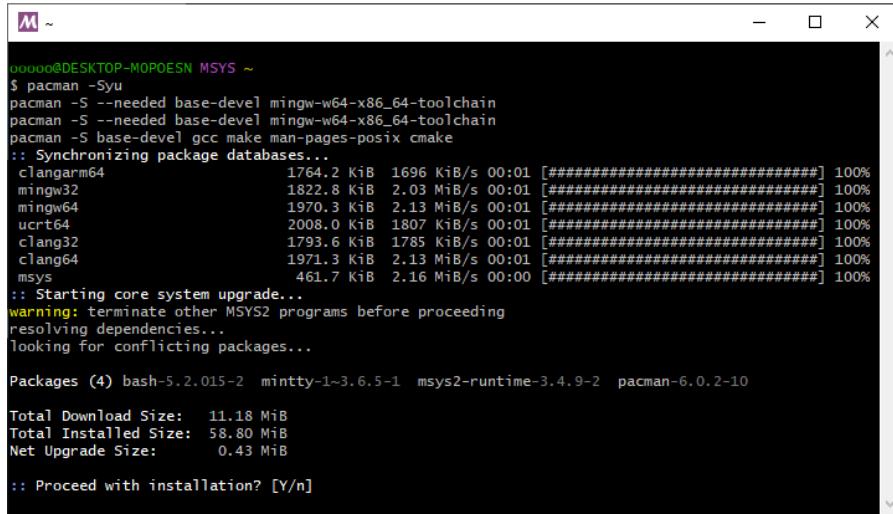


5. Copier et coller les commandes d'installation des paquets vues plus haut :

A screenshot of a terminal window titled "MSYS ~". The window contains the following text:

```
oooooooo@DESKTOP-MOPOESN MSYS ~
$ pacman -Syy
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
pacman -S base-devel gcc make man-pages-posix cmake
```

6. Inscrire une caractère 'y' puis appuyer sur "entrée". MSYS2 devra se fermer et se relancer après sa première mise à niveau, il faudra ouvrir à nouveau le même terminal et relancer les 4 commandes vues plus haut.



```

$ pacman -S --needed base-devel mingw-w64-x86_64-toolchain
pacman -S --needed base-devel mingw-w64-x86_64-toolchain
pacman -S base-devel gcc make man-pages-posix cmake
:: Synchronizing package databases...
clangarm64          1764.2 Kib  1696 KiB/s 00:01 [########################################] 100%
mingw32              1822.8 Kib  2.03 MiB/s 00:01 [########################################] 100%
mingw64              1970.3 Kib  2.13 MiB/s 00:01 [########################################] 100%
ucrt64               2008.0 Kib  1807 KiB/s 00:01 [########################################] 100%
clang32               1793.6 Kib  1785 KiB/s 00:01 [########################################] 100%
clang64               1971.3 Kib  2.13 MiB/s 00:01 [########################################] 100%
msys                 461.7 Kib  2.16 MiB/s 00:00 [########################################] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

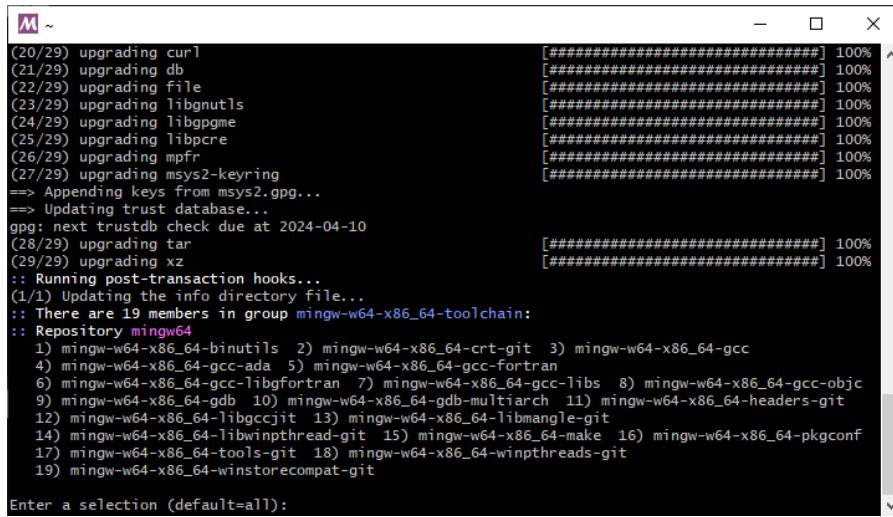
Packages (4) bash-5.2.015-2 mingetty-1~3.6.5-1 msys2-runtime-3.4.9-2 pacman-6.0.2-10

Total Download Size: 11.18 MiB
Total Installed Size: 58.80 MiB
Net Upgrade Size: 0.43 MiB

:: Proceed with installation? [Y/n]

```

7. Lors du choix multiple appuyer sur "entrée" pour installer tous les outils relatifs à MinGW (sinon 3 : gcc sera suffisant en réalité).



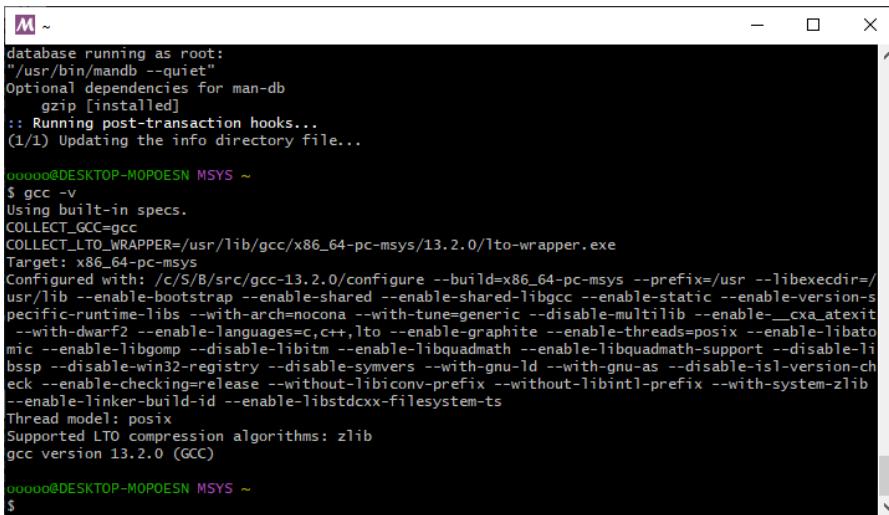
```

(20/29) upgrading curl                               [########################################] 100%
(21/29) upgrading db                                [########################################] 100%
(22/29) upgrading file                             [########################################] 100%
(23/29) upgrading libgnutls                         [########################################] 100%
(24/29) upgrading libpgm                           [########################################] 100%
(25/29) upgrading libpcre                          [########################################] 100%
(26/29) upgrading mpfr                            [########################################] 100%
(27/29) upgrading msys2-keyring                   [########################################] 100%
==> Appending keys from msys2.gpg...
==> Updating trust database...
gpg: next trustdb check due at 2024-04-10
(28/29) upgrading tar                             [########################################] 100%
(29/29) upgrading xz                                [########################################] 100%
:: Running post-transaction hooks...
(1/1) Updating the info directory file...
:: There are 19 members in group mingw-w64-x86_64-toolchain:
:: Repository mingw64
1) mingw-w64-x86_64-binutils 2) mingw-w64-x86_64-crt-git 3) mingw-w64-x86_64-gcc
4) mingw-w64-x86_64-gcc-ada 5) mingw-w64-x86_64-gcc-fortran
6) mingw-w64-x86_64-gcc-libfortran 7) mingw-w64-x86_64-gcc-libs 8) mingw-w64-x86_64-gcc-objc
9) mingw-w64-x86_64-gdb 10) mingw-w64-x86_64-gdb-multiarch 11) mingw-w64-x86_64-headers-git
12) mingw-w64-x86_64-libgccjit 13) mingw-w64-x86_64-libmangle-git
14) mingw-w64-x86_64-libwinpthread-git 15) mingw-w64-x86_64-make 16) mingw-w64-x86_64-pkgconf
17) mingw-w64-x86_64-tools-git 18) mingw-w64-x86_64-winthreads-git
19) mingw-w64-x86_64-winstorecompat-git

Enter a selection (default=all):

```

8. Vérifier l'installation de gcc : dans le cas où ceci afficher autre chose que "gcc non trouvé", il est installé (peu importe la version).



```
M ~
database running as root:
"/usr/bin/man-db --quiet"
Optional dependencies for man-db
    gzip [installed]
:: Running post-transaction hooks...
(1/1) Updating the info directory file...

ooooo@DESKTOP-MOPOESN MSYS ~
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-msys/13.2.0/lto-wrapper.exe
Target: x86_64-pc-msys
Configured with: /c/S/B/src/gcc-13.2.0/configure --build=x86_64-pc-msys --prefix=/usr --libexecdir=/usr/lib --enable-bootstrap --enable-shared --enable-shared-libgcc --enable-static --enable-version-specic-runtime-libs --with-arch=nocona --with-tune=generic --disable-multilib --enable-_cxa_atexit --with-dwarf2 --enable-languages=c,c++,lto --enable-graphite --enable-threads=posix --enable-libatomic --enable-libgomp --disable-libitm --enable-libquadmath --enable-libquadmath-support --disable-libssp --disable-win32-registry --disable-symvers --with-gnu-ld --with-gnu-as --disable-isl-version-check --enable-checking=release --without-libiconv-prefix --without-libintl-prefix --with-system-zlib --enable-linker-build-id --enable-libstdcxx-fsysteem-ts
Thread model: posix
Supported LTO compression algorithms: zlib
gcc version 13.2.0 (GCC)
ooooo@DESKTOP-MOPOESN MSYS ~
$
```

1.5.2 Compilation avec gcc

Il est possible d'obtenir un exécutable avec `gcc` par la commande

```
gcc -o [APPLICATION] [FICHIERS SOURCES]
```

où `[APPLICATION]` est remplacé par le nom que l'on souhaite donner à l'application et `[FICHIERS SOURCES]` est la liste des fichiers source séparés deux à deux par un espace. Pour compiler et linker notre fichier `hello_esgi.c` en `SuperApp`, dans le répertoire où se trouve le fichier, il faut saisir :

```
gcc -o SuperApp hello_esgi.c
```

`SuperApp` pourra ensuite se lancer par la commande suivante :

```
./SuperApp
```

Ceci donne la réponse suivante dans le terminal :

```
Hello ESGI !
```

le `.`/`/` est nécessaire pour dire que l'on commence de l'emplacement courant dans le système de fichier, sinon le système d'exploitation cherchera dans ses commandes et applications installées.

Dans la suite du cours, votre professeur supposera que vous travaillez avec gcc sous MSYS2 64 bits, sous un environnement Linux 64 bits ou équivalent.

1.5.3 Utiliser gcc sous Windows

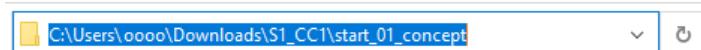
Via MSYS2

Une fois gcc installé via MSYS2, la compilation peut se faire sous Windows en suivant les instructions suivantes :

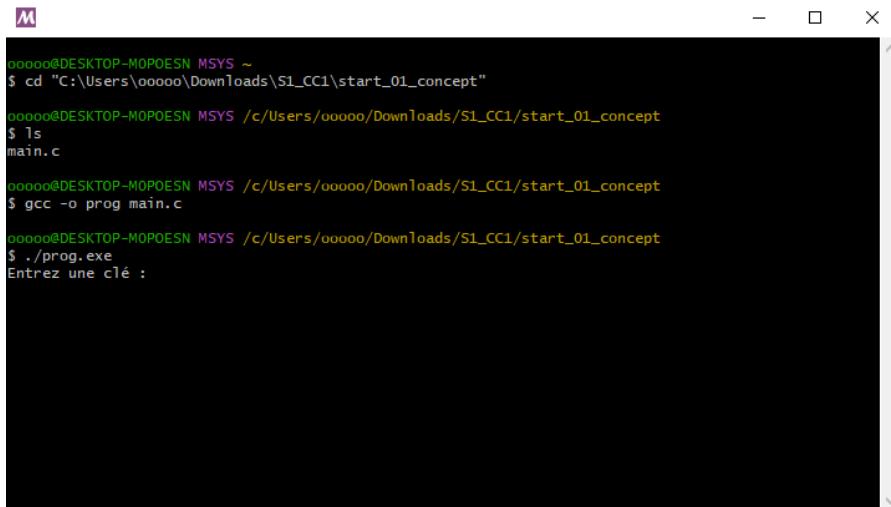
1. Ouvrir un terminal MSYS2 :



2. La manière la plus simple de se rendre là où existe votre fichier `.c` est d'utiliser la commande `cd` (change directory), par défaut un terminal MSYS2 s'ouvre à l'emplacement `C:\msys64\home\[user]`. Pour ceci, copiez le chemin vers le dossier où se trouve votre fichier depuis l'explorateur et copiez le entre guillemets.



3. Une fois le répertoire ciblé, vous pourrez compiler et exécuter votre programme comme vu précédemment :



```
ooooo@DESKTOP-MOPOESN MSYS ~
$ cd "C:\Users\ooooo\Downloads\S1_CC1\start_01_concept"
ooooo@DESKTOP-MOPOESN MSYS /c/Users/ooooo/Downloads/S1_CC1/start_01_concept
$ ls
main.c

ooooo@DESKTOP-MOPOESN MSYS /c/Users/ooooo/Downloads/S1_CC1/start_01_concept
$ gcc -o prog main.c
ooooo@DESKTOP-MOPOESN MSYS /c/Users/ooooo/Downloads/S1_CC1/start_01_concept
$ ./prog.exe
Entrez une clé :
```

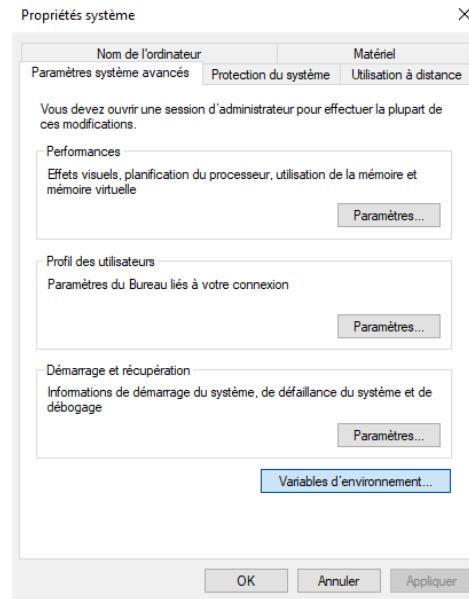
4. Notez que vous pourriez vouloir proposer votre programme à un utilisateur, pour ceci vous pouvez copier le fichier `msys-2.0.dll` du dossier `C:\msys64\usr\bin` à côté de votre exécutable :

Nom	Modifié le	Type	Taille
main	24/10/2023 17:30	Fichier C	1 Ko
msys-2.0.dll	11/01/2023 21:10	Extension de l'app...	23 162 Ko
prog	24/10/2023 17:43	Application	29 Ko

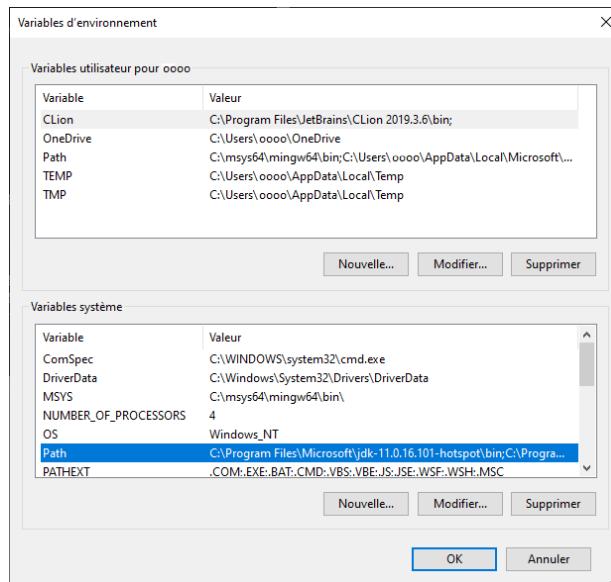
Ailleurs : variables d'environnement

Vous pourriez vouloir utiliser des commandes installées via MSYS2 en dehors de son environnement (comme via votre IDE ou votre invite de commandes Windows). Pour ceci vous devrez ajouter le répertoire `bin` où sont installés les exécutables utilisés par MSYS2 dans la variable d'environnement système PATH :

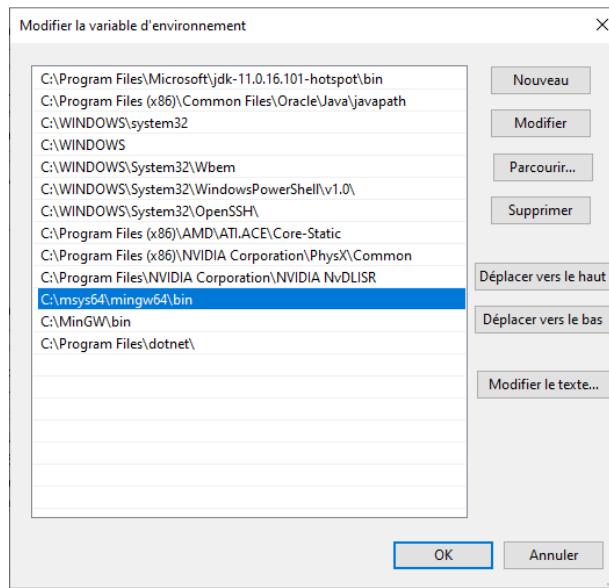
1. Rechercher dans les paramètres l'édition des variables d'environnement :



2. Choisir la variable d'environnement système PATH :



3. Ajouter le chemin C:\msys64\mingw64\bin (note : si d'autres installation existent, vous pouvez prioriser MSYS2 en la plaçant plus haut dans la liste) :



4. gcc est maintenant disponible via votre invite de commandes et ailleurs dans votre système :

```

Microsoft Windows [version 10.0.19045.3448]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\ooooo>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/13.2.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../gcc-13.2.0/configure --prefix=/mingw64 --with-local-prefix=/mingw64/local --build=x86_64-w64-mingw32
--host=x86_64-w64-mingw32 --target=x86_64-w64-mingw32 --with-native-system-header-dir=/mingw64/include --libexecdir=/mingw64/lib
--enable-bootstrap --enable-checking=release --with-arch=none --with-tune=generic --enable-languages=c,lto,c++
,fortran,ada,obje,obj-c++,jil --enable-shared --enable-static --enable-libatomic --enable-threads=posix --enable-graph
ite --enable-fully-dynamic-string --enable-libstdcxx-filesystem-ts --enable-libstdcxx-time --disable-libstdcxx-pch --ena
ble-lto --enable-libgom --disable-libssp --disable-multilib --disable-rpath --disable-win32-registry --disable-nls --di
sable-werror --disable-symvers --with-libiconv --with-system-zlib --with-gmp=/mingw64 --with-mpfr=/mingw64 --with-mpc=/m
ingw64 --with-isl=/mingw64 --with-pkgversion='Rev2, Built by MSYS2 project' --with-bugurl=https://github.com/msys2/MINGW
/packages/issues --with-gnu-as --with-gnu-ld --disable-libstdcxx-debug --with-boot-ldflags=-static-libstdc++ --with-stag
el-ldflags=-static-libstdc++
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 13.2.0 (Rev2, Built by MSYS2 project)

C:\Users\ooooo>

```

1.5.4 Mac alternative : Xcode

Mac peut avoir le compilateur clang en alternative à gcc. Sinon vous pouvez aussi installer Xcode pour développer en langage C sous Mac. Un tutoriel est dis-

ponible au lien suivant :

<https://www.cs.auckland.ac.nz/~paul/C/Mac/xcode/>.

Notez que le téléchargement de Xcode peut prendre un certain temps.

1.5.5 Online GDB : compilateur en ligne

Si aucune des solutions proposées précédemment ne vous convient, l'installation d'outils sur votre ordinateur personnel vous dérange, vous avez un Chrome-Book ou que vous n'avez pas les droits pour installer les outils nécessaires, vous avez toujours la possibilité de travailler en langage C à l'aide d'un compilateur en ligne : [Online GDB](#).

Notez que cet environnement fonctionne sous une machine Linux 64 bits et convient pour suivre ce cours (sauf pour les mini-projets où la bibliothèque SDL devrait être utilisée).

1.6 Lire la documentation

Lire une documentation est malheureusement une tâche récurrente pour une programmeur. Bien que beaucoup s'essaient à imaginer savoir comment fonctionne un code qu'ils n'ont pas codé ou demandent sans rechercher, ce qui lui vaudra probablement l'injonction "RTFM" lorsqu'il s'étonnera de ne pas s'en sortir.

1.6.1 Dans son terminal

Sous Linux, nous l'avions installé via le paquet `manpages-dev` et sous Windows via `man-pages-posix`. Vous pouvez accéder au manuel d'une fonctionnalité en langage C via `man 3 [fonction]`. Par exemple, pour se renseigner sur `printf` :

```
man 3 printf
```

```
PRINTF(3)          Linux Programmer's Manual          PRINTF(3)

NAME
    printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf,
    vdprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int dprintf(int fd, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);

#include <stdarg.h>

int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vdprintf(int fd, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);

Manual page printf(3) line 1 (press h for help or q to quit)
```

1.6.2 En ligne sur [cppreference.com](#)

Une bonne documentation (pour le Langage C++ et) pour le Langage C est le site [cppreference.com](#).

Bien que par défaut en anglais, vous pouvez y trouver :

- une référence sur le Langage C en français : <https://fr.cppreference.com/w/c>
- les fonctions des fichiers d'entête (comme `stdio` par exemple) :
<https://fr.cppreference.com/w/c/header>

Nous vous donnons des éléments de base pour débuter via ce cours, mais n'hésitez pas à compléter vos connaissances avec la documentation. De nombreuses erreurs pourraient être dues à une lecture incomplète d'une documentation. Vos outils prêts, vous êtes donc en passe de commencer votre initiation à la programmation via le Langage C ! Courage et accrochez vous !

1.7 Résumé

Quelques mots-clés :

Algorithme : Procédé automatique qui depuis une entrée opère dessus pour obtenir une sortie souhaitée.

Instruction : Ordre donné à un ordinateur (en langage C, c'est l'ensemble de caractères précédant un point-virgule).

Programme : Idée conceptuelle d'instructions ordonnées qui aboutissent à un comportement souhaité.

Langage de programmation : Ensemble de mots clés qui permettent l'écriture d'instructions.

Code source : Suite d'instructions données dans un langage de programmation qui définissent un programme.

Binaire : Représentation d'information comme une suite de 0 et de 1.

Langage machine : Suite de valeurs binaires qui indiquent à la machine les actions à effectuer sur les composants de l'ordinateur.

Exécutable / Application : fichier donné en langage machine qui réalise un programme.

Compilation : Étape de transformation en langage machine d'un code source donné.

Linkage : Édition des liens entre différentes parties d'un code source compilé menant à la création d'un exécutable.

Compiler un code source en langage C donné par des fichiers [FICHIERS SOURCES] en une application [APPLICATION] :

```
gcc -o [APPLICATION] [FICHIERS SOURCES]
```

main indique au programme où commencer à lire les instructions lors du lancement de l'application.

Ajouter les fonctionnalités d'une bibliothèque C [BIBLIOTHÈQUE] pour les utiliser dans le code source :

```
#include <[BIBLIOTHÈQUE]>
```

Commande issue de `stdio.h` (standard d'entrées et sorties) pour envoyer du texte pour impression dans le terminal :

```
printf(" [TEXTE] ");
```

Terminer l'exécution de l'application :

```
exit(EXIT_SUCCESS);
```

1.8 Entrainement

Exercice 3 (* Votre Hello ESGI).

1. Créez un fichier `ESGI_1_TP_1_NOM_Prenom.c` en remplaçant NOM et Prenom par les vôtres.
2. Écrivez un programme qui affiche la sortie suivante (en remplaçant NOM et Prenom par les vôtres) :

```
ESGI : Bachelor première année  
Seance 1  
NOM Prenom
```

3. Compilez et linkez le programme pour obtenir un exécutable (peu importe le système d'exploitation).
4. Lancez votre programme pour admirer le résultat !

2 Variables

2.1 Introduction et motivation

Nous avons vu précédemment un programme qui affiche du texte pour communiquer avec l'utilisateur via une console. Il est possible de donner plus de vie à un programme pour prendre en compte et traiter des données communiquées par l'utilisateur. Pour ceci, je propose de tester le code suivant :

ask_age.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int age; /* Déclare une variable */
6     printf("Quel est votre age ? "); /* Affiche du texte
7         */
7     scanf("%d", &age); /* Lit une valeur et la met dans
8         'age' */
8     printf("Vous avez donc %d ans !\n", age); /* Affiche
9         la valeur de 'age' */
9     exit(EXIT_SUCCESS);
10 }
```

Après compilation et exécution, ceci donne par exemple la sortie suivante :

```
# gcc -ansi -Wall -o ask_age ask_age.c
# ./ask_age
Quel est votre age ? 42
Vous avez donc 42 ans !
```

Notons qu'un code en C permet de donner des instruction à la machine, mais aussi de communiquer avec des humains. N'importe qui peut être amené à relire votre code, y compris vous-même (aujourd'hui ou dans quelques temps), pensez donc à laisser des commentaires pour faciliter cette lecture. Un commentaire est une suite de caractères ignorés de la machine et se déclare entre les symboles /* et */ :

Exemple de commentaire

```
1  /* Un commentaire qui ne semble pas utile mais qui */
2  /* permettra une lecture avec moins de douleur */
3  /* pour mon professeur et moi dans quelques années.*/
```

Nous avons dans un premier temps demandé à avoir un **variable** age dans laquelle ranger un nombre entier. C'est comme avoir un assistant qui traite un ensemble de boîte où ranger des informations. On lui demande au préalable une boîte à laquelle on donne un nom pour pouvoir la redemander plus tard et l'utiliser.

```
int age;
```

À l'aide de **printf** nous imprimons les caractères à envoyer à l'utilisateur pour lui demander son âge. C'est ensuite **scanf** qui permet de scanner la saisie de l'utilisateur pour la traiter.

```
printf("Quel est votre age ? ");
scanf("%d", &age);
```

Une fois l'information lue et traitée, nous pouvons répondre en partageant l'information récupérée. Donc si l'utilisateur saisit '42', nous enregistrons cette information dans **age** puis nous pouvons communiquer la valeur de **age** qui est '42'.

```
printf("Vous avez donc %d ans !\n", age);
```

À ce point, certaines subtilités sont à remarquer et vont être explicitées ensuite :

1. **age** est précédée d'un mot clé **int** à la ligne 5 (explication en section 2.2).

2. la suite de caractères %d est présente pour `printf` et `scanf` aux lignes 7 et 8 (explication en section [2.3](#)).
3. `age` est précédée du symbole & dans `scanf` et non dans `printf` (explication en section [2.4](#)).

Une autre subtilité intéressante à remarquer est que les options `-ansi` `-Wall` ont été ajoutées lors de la compilation avec `gcc`. Ceci indique que l'on souhaite respecter la norme ANSI et `-Wall` donne des avertissements pour aider à produire un meilleur code. Ce code peut encore être optimisé en ajoutant l'option `-O2` voir `-O3` pour améliorer les performances du programme dès la compilation.

2.2 Variables typées

2.2.1 Déclaration

En langage C, une variable possède un type qu'il faut indiquer à l'avance à la machine pour qu'elle sache l'espace mémoire à réserver pour la variable. Ce type permet principalement de paramétriser la taille en mémoire de la variable et choisir de gérer des nombres entiers (une boîte avec un nombre de billes) ou des nombres à virgule flottante (un bidon avec une quantité d'eau).

Un nom de variable ne doit pas commencer par un chiffre et peut être composé de lettres, chiffres et de tiret-bas. Par convention un nom de variable commencera par une lettre minuscule. Veillez à bien nommer vos variables, ça remplace des commentaire et augmente la lisibilité de votre code.

Une telle déclaration se fait sous la forme suivante :

```
type nom_de_variable;
type nom_de_variable = VALEUR;
type nom_de_variable1 = VALEUR1, nom_de_variable2 = VALEUR2;
```

D'où l'exemple, vu précédemment nous déclarions une variable avec pour `nom_de_variable` : `age` de type : `int`.

```
int age;
```

À noter que par défaut une telle déclaration ne permet pas de connaître la valeur de la variable (ceci dépend de ce qui a été écrit dans la mémoire précédemment). Il est donc possible et souvent préférable d'indiquer une valeur par défaut

pour cette variable avant de l'utiliser pour avoir plus de contrôle sur son code. Par exemple :

```
int reponse = 42;
```

Il est aussi possible de déclarer plusieurs variables de même type sur une même ligne en séparant leur déclaration par une virgule :

```
int first = 1, second = 2, third = 3;
```

Si plus tard la variable devait être amenée à changer de valeur, il est possible de lui affecter une nouvelle valeur en utilisant le symbole '`=`'. À bien noter que ce symbole comme précédemment dit 'mettre la valeur de droite à l'emplacement donné à gauche'.

```
reponse = 1337;
```

2.2.2 Entiers signés

Le type `int` indique que l'on souhaite manipuler cette variable comme un nombre entier. Il existe cependant d'autres types d'entiers signés avec plus ou moins de contenance :

Type	Espace mémoire	Borne inférieure	Borne supérieure
<code>char</code>	1 octet	-128	127
<code>short</code>	2 octets	-32 768	32 767
<code>int</code>	4 octets	-2 147 483 648	2 147 483 647
<code>long</code>	8 octets	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

À noter que le type `char` est généralement utilisé pour sauvegarder un caractère ASCII. En C, il existe une représentation de caractères en ASCII donnés entre simple apostrophes qui correspondent à la valeur numérique qui les représente :

```
char zero = '0'; /* vaut 48 */
char lettre_majuscule = 'A'; /* vaut 65 */
char lettre_minuscule = 'a'; /* vaut 97 */
```

À noter que sur la machine, la mémoire est une longue bande de 0 et de 1 que l'on nomme de bits (binary digits). On considère que nous pouvons représenter cette mémoire par groupes de 8 bits qui correspondent chacun à un octet. Donc 2 octets c'est 16 bits, 4 octets c'est 32 bits et ainsi de suite.

2.2.3 Entiers non-signés

Les types `char`, `short`, `int`, `long` sont des types **signés** ceci veut dire que ce sont des entiers qui admettent un signe et donc qui peuvent être négatifs. Il existe aussi la possibilité de déclarer des entiers non signes en précisant le mot-clé `unsigned` devant le type d'entier :

Type	Espace mémoire	Borne inférieure	Borne supérieure
<code>unsigned char</code>	1 octet	0	255
<code>unsigned short</code>	2 octets	0	65 535
<code>unsigned int</code>	4 octets	0	4 294 967 295
<code>unsigned long</code>	8 octets	0	18 446 744 073 709 551 615

2.2.4 Nombres à virgule flottante

Cependant les entiers ne permettent pas de représenter directement des nombres à virgule, pour ceci, nous avons des nombres flottants donnés par le type `float`. Un nombre flottant est en réalité une approximation (la mémoire de la machine n'étant pas infinie). Pour améliorer la plage de valeurs que peut prendre cette approximation et la précision de l'approximation, il existe aussi le type `double` qui occupe deux fois la taille d'un `float` comme donné dans la table suivante :

Type	Espace mémoire	Borne inférieure	Borne supérieure
<code>float</code>	4 octets	-3,402 823 .10 ³⁸	3,402 823 .10 ³⁸
<code>double</code>	8 octets	-1,797 693 .10 ³⁰⁸	1,797 693 .10 ³⁰⁸
<code>long double</code>	16 octets	-1,189 731 .10 ^{4 932}	1,189 731 .10 ^{4 932}

Pour affecter une valeur flottante à une variable, la notation à virgule est correspond à la notation anglaise : la virgule devient un point et il faut au moins un chiffre de l'un des côtés du point (le reste est considéré rempli de zéros). Le point n'est pas obligatoire si la valeur est entière. Ceci donne par exemple l'affectation suivante :

```
float valeur_flotante = 0.42;
valeur_flotante = .42;
valeur_flotante = 1337.;
valeur_flotante = 1235;
```

Il est aussi possible de donner une valeur en notation scientifique à l'aide d'une syntaxe [MATISSE]e[EXPOSANT]. Ceci permet par exemple d'écrire 42 000 = 42.10³ comme 42.e3 :

```
float valeur_flotante = 42.e3;
```

Une subtilité est que dans le cas de `long double`, pour des valeurs qui ne tiennent pas sur un `double`, il faudra préciser un 1 derrière la valeur pour indiquer que l'on souhaite construire un `long double` (par défaut les valeurs flottantes sont des `double`).

```
long double giant = 1e2048l;
```

2.2.5 Constantes

Il est possible dans un programme de devoir réutiliser des valeurs constantes que l'on définit une unique fois. Pour éviter de devoir changer cette valeur arbitraire à chaque fois qu'on la trouve dans un programme, il est possible de définir une valeur constante à l'aide du mot-clé `const` pour ne la changer qu'en haut du programme. Cependant le programme ne pourra pas changer cette valeur pendant l'exécution.

```
const int reponse_inalienable = 42;
```

Exercice 4 (★ Déclarer et définir des variables).

Compléter le programme suivant en fonction de ce qui est indiqué en commentaire :

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    /* TODO : déclarer un entier ent1 */
    /* TODO : déclarer un nombre à virgule vir1 */
    /* TODO : définir un entier ent2 à valeur 42 */
    /* TODO : définir un entier chr1 à valeur '@' */
    /* TODO : faire prendre à vir1 la valeur 13.37 */
    /* TODO : faire prendre à ent1 la valeur de ent2 */

    exit(EXIT_SUCCESS);
}
```

2.3 Format d'affichage

2.3.1 Printf

Nous avons vu précédemment que la procédure `printf` de la bibliothèque `stdio.h` permet d'imprimer des caractères dans la console pour communiquer avec l'utilisateur. Cette procédure permet aussi d'imprimer les valeurs de nos variables à l'écran.

`printf` demande dans un premier temps une chaîne de caractère (texte donnée entre guillemets). Dans cette chaîne de caractères, il est possible de préciser des parties que l'on voudra remplacer par des valeurs données par le programme. Ces précisions se font par le caractère `%` et ensuite on donne le format qui indique comme lire et écrire la valeur. Pour imprimer le caractère `'%`, il faut le doubler : `%%`. Pour chaque format, `printf` attendra une valeur dans la liste qui suit la chaîne de caractères.

Dans notre exemple, nous avions utilisé `%d`, ceci permet d'afficher des entiers `char`, `short`, `int`. Dans le même esprit, il est possible de proposer l'exemple suivant :

```
printf("%d + %d = %d\n", 1, 1, 2);
```

Ce qui affichera dans la sortie de la console la ligne suivante :

```
1 + 1 = 2
```

2.3.2 Formats pour entiers

Nous avons aussi vu qu'un caractère est représenté par un entier. Il est naturellement possible d'afficher le nombre associé au caractère par `%d` mais aussi d'afficher le caractère représenté par `%c` :

```
char chiffre = '2';
int lettre = 'x';
printf("chiffre = %d : '%c'\n", chiffre, chiffre);
printf("lettre = %d : '%c'\n", lettre, lettre);
```

En sortie :

```
chiffre = 50 : '2'
lettre = 120 : 'x'
```

Notons que `%d` ne peut afficher que des entiers `int` et de plus petite contenance. Pour afficher des `long`, il sera nécessaire d'ajouter un `l` dans le format pour obtenir le format `%ld` et préciser cette différence.

```
long big = 50000000000000;
printf("big est un entier : %d\n", big);
printf("oups, big porte bien son nom : %ld\n", big);
```

En sortie :

```
big est un entier : 658067456
oups, big porte bien son nom : 50000000000000
```

À noter que le compilateur peut nous avertir à l'aide d'un warning précisant que l'on affiche un '`long int`' (`long`) comme un `int` :

```
warning: format '%d' expects argument of type 'int', but argument
→ 2 has type 'long int' [-Wformat=]
    printf("big est un entier : %d\n", big);
          ^ ^
          %ld
```

Nous avions aussi vu qu'en réalité les nombres sont représentés sous forme binaire en machine. Un format souvent présenté pour représenter le binaire sur des octets est la notation hexadécimale. Pour comprendre cette subtilité, il faut noter que nous représentons nos nombres en base 10, ce qui veut dire que nous avons des chiffres allant de 0 à 9 que nous regroupons devant une unité, dizaine, centaine, puissance de 10 pour former notre nombre :

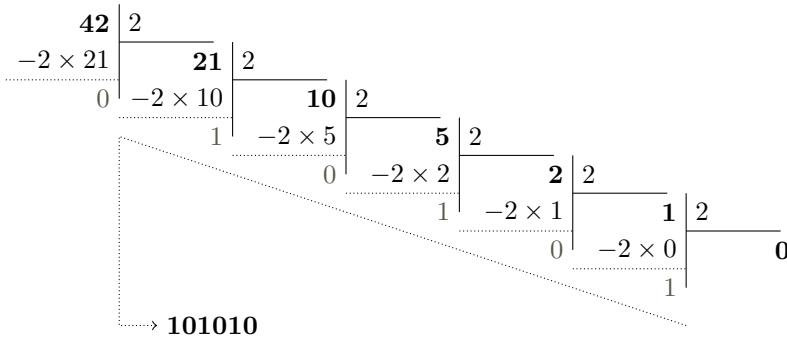
$$42 = 4 \times 10 + 2 \times 1 = 4 \times 10^1 + 2 \times 10^0$$

Pour comprendre la notation binaire, c'est considérer qu'au lieu de prendre une base 10, nous prenons une base 2 et donc les chiffres sont 0 et 1 en notation binaire.

D'où :

$$\begin{aligned}
 42 &= 32 + 8 + 2 \\
 &= \mathbf{1} \times 32 + \mathbf{0} \times 16 + \mathbf{1} \times 8 + \mathbf{0} \times 4 + \mathbf{1} \times 2 + \mathbf{0} \times 1 \\
 &= \mathbf{1} \times 2^5 + \mathbf{0} \times 2^4 + \mathbf{1} \times 2^3 + \mathbf{0} \times 2^2 + \mathbf{1} \times 2^1 + \mathbf{0} \times 2^0 \\
 42 &= (101010)_2
 \end{aligned}$$

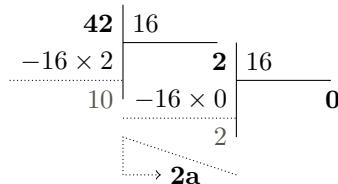
Ceci peut aussi se calculer par des division euclidiennes successives par 2 :



La notation hexadécimale, c'est une notation en base 16 où on étend les chiffres par les lettres : 'a' = 'A' = 10, 'b' = 'B' = 11, 'c' = 'C' = 12, 'd' = 'D' = 13, 'e' = 'E' = 14, 'f' = 'F' = 15. Par exemple :

$$\begin{aligned}
 42 &= 32 + 8 \\
 &= \mathbf{2} \times 16 + \mathbf{10} \times 1 \\
 &= \mathbf{2} \times 16^1 + \text{'a'} \times 16^0 \\
 42 &= (2a)_{16}
 \end{aligned}$$

Ceci peut aussi se calculer par des division euclidiennes successives par 16 :



Le passage de la notation binaire à hexadécimale se fait en regroupant par 4 les bits ce qui est équivalent à un chiffre en hexadécimal comme illustré dans la table de correspondances suivante :

Décimal :	0	1	2	3	4	5	6	7
Hexadécimal :	0	1	2	3	4	5	6	7
Binaire :	0000	0001	0010	0011	0100	0101	0110	0111
Décimal :	8	9	10	11	12	13	14	15
Hexadécimal :	8	9	a	b	c	d	e	f
Binaire :	1000	1001	1010	1011	1100	1101	1110	1111

Pour utiliser cette représentation en langage C, on fait précéder le nombre écrit en hexadécimal par `0x` pour définir un entier et on utilise le format `%x` (minuscules) ou `%X` (majuscules) pour lire le nombre en hexadécimal :

```
int reponse = 0x2a;
printf("La reponse a la vie est %x !\n", reponse);
printf("Ah, pour les humains ? C'est %d !\n", reponse);
```

En sortie :

```
La reponse a la vie est 2a !
Ah, pour les humains ? C'est 42 !
```

Si l'entier n'est pas assez long, il est possible de combler d'espaces les caractères où il manque des chiffres en précisant soit le nombre d'espaces à ajouter après le `%` pour avoir un nombre de chiffre attendus au minimum :

```
printf("%7d\n", 1);
printf("%7d\n", 1000);
printf("%7d\n", 1000000);
```

En sortie :

```
1
1000
1000000
```

Le même, si on ajoute un 0 avant ce nombre, ceci comblera de zéros au lieu d'espaces. Illustrons cet exemple pour visualiser le plus grand entier que l'on peut représenter (en non-signé avec le format %u) :

```
unsigned long giant = 0xffffffffffffffff;
printf("int : 0x%016x (%u)\n", giant, giant);
printf("long : 0x%016lx (%lu)\n", giant, giant);
```

En sortie :

```
int : 0x00000000ffffffff (4294967295)
long : 0xffffffffffffffff (18446744073709551615)
```

2.3.3 Formats pour flottants

Le format d'affichage des nombres flottants est différent de celui des entiers. En effet le format d'affichage se base sur la représentation mémoire de ce qu'on lui donne, celle est flottants est différente de celle des entiers, ce qui demande l'utilisation d'autres formats :

- %f affiche un flottant sous forme numérique.
- %e affiche un flottant en notation scientifique.
- %g propose un affichage automatique (choix de la précision à afficher et passage au besoin en notation scientifique).

```
float v = 12345.6789;
printf("Avec %f : %f\n", v);
printf("Avec %e : %e\n", v);
printf("Avec %g : %g\n", v);
```

En sortie :

```
Avec %f : 12345.678711
Avec %e : 1.234568e+04
Avec %g : 12345.7
```

Avec un nombre plus grand, ceci donnerait :

```
float v = 1e30;  
printf("Avec %f : %f\n", v);  
printf("Avec %e : %e\n", v);  
printf("Avec %g : %g\n", v);
```

En sortie :

```
Avec %f : 1000000015047466219876688855040.000000
Avec %e : 1.000000e+30
Avec %g : 1e+30
```

Sur le dernier exemple, nous pouvons observer les problèmes de précision dues à l'approximation. Pour ceci nous avons à disposition les types `double` et `long double`. Pour l'affichage d'un `double`, ceci peut se faire comme pour un `float`. Cependant pour l'affichage d'un `long double`, ceci nécessitera l'ajout d'un L dans le format :

```
float un_float = 1e30f;
double un_double = 1e30;
long double un_long_double = 1e30l;
printf("float      : %f\n", un_float);
printf("double     : %f\n", un_double);
printf("long double : %Lf\n", un_long_double);
```

En sortie :

```
float      : 1000000015047466219876688855040.000000
double     : 1000000000000000000019884624838656.000000
long double : 10000000000000000000024696061952.000000
```

Il est aussi possible de régler la précision de l'affichage d'un flottant en paramétrant le nombre de chiffres après la virgule en ajoutant un point et la précision d'affichage attendue :

```
long double pi =
    ↵ 3.1415926535897932384626433832795028841971693993751058201;
printf("pi vaut environ %.2Lf\n", pi);
printf("Pas assez précis ? C'est aussi %.16Lf\n", pi);
```

En sortie :

```
pi vaut environ 3.14
Pas assez précis ? C'est aussi 3.1415926535897932
```

Exercice 5 (★ Afficher des variables).

Compléter et modifier si besoin le programme suivant en fonction de ce qui est indiqué en commentaire :

```
#include <stdio.h>
#include <stdlib.h>

int main() {

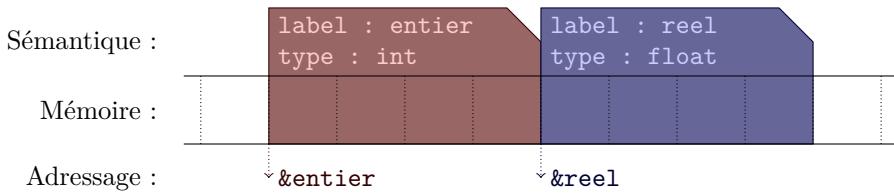
    float reel = 13.37;
    char caractere = 'Z';
    int entier = 42000000000;

    /* TODO : afficher reel */
    /* TODO : afficher caractere */
    /* TODO : afficher caractere en décimal */
    /* TODO : afficher entier : 4200000000 */
    /* TODO : afficher entier en hexadécimal */

    exit(EXIT_SUCCESS);
}
```

2.4 Adresse d'une variable

Nous avons dit que la mémoire est une longue bande valeurs binaires. Pour certaines utilisations, il sera nécessaire de savoir où une variable commence sur cette bande. Pour ceci chaque variable a une adresse que la machine lui a attribué à la création de la variable et que l'on peut lui demander. Un peu comme quand on vous demande l'adresse de votre résidence. Cette adresse s'obtient en préfixant le nom de la variable par `&`.



Il est possible d'afficher l'adresse d'une variable avec `%p` :

```
int variable = 42;
printf("Valeur de variable : %d\n", variable);
printf("Adresse de variable : %p\n", &variable);
```

En sortie :

```
Valeur de variable : 42
Adresse de variable : 0x7ffdafbec408
```

2.5 Sccanf

Cette adresse est utile pour la procédure `scanf` qui lit une valeur saisie par l'utilisateur puis la range à un endroit demandé : d'où l'adresse de la variable pour lui livrer la valeur lue.

`scanf` fonctionne dans la même idée que `printf` : on fournit un format puis la liste des emplacements qui recevront les valeurs lues.

```
int variable = 42;
printf("variable vaut %d\n", variable);
printf("Entrez une nouvelle valeur pour variable : ");
scanf("%d", &variable);
printf("variable vaut maintenant %d\n", variable);
```

En sortie :

```
variable vaut 42
Entrez une nouvelle valeur pour variable : 1337
variable vaut maintenant 1337
```

Nous retrouvons globalement le même principe de formatage qu'avec `printf` :

```
char caractere;
printf("Entrez un caractere : ");
scanf("%c", &caractere);
printf("Voici votre caractere : '%c'\n", caractere);
```

En sortie :

```
Entrez un caractere : #
Voici votre caractere : '#'
```

Pour les flottants, le seul format à garder en tête est `%f`, mais une subtilité apparaît pour gérer les `double` qui réclament un `%lf` et `long double` reste valable avec `%Lf`.

```
float petit_flottant;
double moyen_flottant;
long double grand_flottant;
printf("Entrez trois valeurs flottantes :\n");
scanf("%f %lf %Lf", &petit_flottant, &moyen_flottant,
      &grand_flottant);
printf("Voici vos valeurs :\n * %g\n * %g\n * %Lg\n",
      petit_flottant, moyen_flottant, grand_flottant);
```

En sortie :

```
Entrez trois valeurs flottantes :
3.1415 1.e-6 42.e1337
Voici vos valeurs :
* 3.1415
* 1e-06
* 4.2e+1338
```

Exercice 6 (★★ Lire et afficher une variable).

Compléter le programme suivant en fonction de ce qui est indiqué en commentaire :

```
#include <stdio.h>
#include <stdlib.h>

int main() {

    double reel;

    /* TODO : demander lecture de reel */
    /* TODO : lire reel */
    /* TODO : afficher reel */

    exit(EXIT_SUCCESS);
}
```

2.6 Résumé

Écrire un commentaire :

```
/* [COMMENTAIRE] */
```

Les entiers signés :

```
char caractere; /*  
de -128  
à 127  
*/  
short entier_court; /*  
de -32 768  
à 32 767  
*/  
int entier; /*  
de -2 147 483 648  
à 2 147 483 647  
*/  
long entier_long; /*  
de -9 223 372 036 854 775 808  
à 9 223 372 036 854 775 807  
*/
```

Les entiers non-signés :

```
unsigned char octet; /*  
de 0  
à 255  
*/  
unsigned short entier_naturel_court; /*  
de 0  
à 65 535  
*/  
unsigned int entier_naturel; /*  
de 0  
à 4 294 967 295  
*/  
unsigned long entier_naturel_long; /*  
de 0
```

```
à 18 446 744 073 709 551 615
*/
```

Les flottants :

```
float flottant; /*
de -3.402 823 e 38
à 3.402 823 e 38
*/
double grand_flottant; /*
de -1.797 693 e 308
à 1.797 693 e 308
*/
long double enorme_flottant; /*
de -1.189 731 e 4 932
à 1.189 731 e 4 932
*/
```

Déclarations plus évoluées :

```
type variable = [VALEUR_PAR DEFAUT];
type variable_1, variable_2;
```

Changer la valeur d'une variable :

```
variable = [VALEUR];
```

Constante (même valeur pour toute durée du programme) :

```
const type variable = [VALEUR];
```

Exemples de saisie de valeurs par défaut :

```
char caractere = 'A';
int entier = 42;
float pi = 3.141;
float scientifique = 1.414e20;
long double enorme_flottant = 1.1414e201;
```

Adresse d'une variable :

```
&variable;
```

Principaux formats pour **printf** (impression terminal) :

```
/* char */
printf("%c %d\n", 'A', 'A');
/* short, int */
printf("%d %x %04d '%4d'\n", 42, 0x2a, 42, 42);
/* long */
printf("%ld %lx\n", 42, 0x2a);
/* entier : unsigned */
printf("%u %x\n", 42, 0x2a);
/* float, double */
printf("%f %g %e %.3f\n", 42., 42., 42., 42.);
/* long double */
printf("%Lf %Lg %Le\n", 42.1, 42.1, 42.1);
/* adresse */
printf("%p\n", &variable);
```

Principaux formats pour **scanf** (lecture terminal) :

```
/* char */
scanf("%c %d", &caractere, &caractere);
/* short, int */
scanf("%d %x", &entier, &entier);
/* long */
scanf("%ld %lx", &entier_long, &entier_long);
/* entier : unsigned */
scanf("%u %x", &entier_naturel, &entier_naturel);
/* float */
scanf("%f", &flottant);
/* double */
scanf("%lf", &grand_flottant);
/* long double */
scanf("%Lf", &enorme_flottant);
```

2.7 Entrainement

Exercice 7 (** Affichage formaté hexadécimal).

Vous devez proposer un programme demandant la lecture d'un entier 64 bits, puis afficher sa représentation en hexadécimal sur 8 octets. Ceci pourrait donner la sortie illustrée par les exemples suivants :

```
Entrez un entier : 42
42 = 0x000000000000002A
```

```
Entrez un entier : 18000000000000000000000000
18000000000000000000000000 = 0xF9CCD8A1C5080000
```

Exercice 8 (* Affectations).

L'ordinateur de votre binôme de projet vient de tomber en panne, mais il a heureusement laissé des instructions pour terminer sa partie, à vous de jouer.

01_binome.c : Code de votre binôme

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char car;
    /* TODO : créer un entier ent */
    /* TODO : mettre '4' dans car */
    /* TODO : mettre 2 dans ent */
    /* TODO : afficher le caractère car */
    /* TODO : afficher ent en suite du précédent affichage
       ↵ */
    /* TODO : afficher l'entier correspondant à car */
    exit(EXIT_SUCCESS);
}
```

Des camarades d'une autre classe : Alice, Bob et Charlie ont chacun commencé leur projet de jeu vidéo révolutionnaire en langage C, mais leur code ne fonctionne pas, aider chacun de vos camarades en leur laissant des commentaires pour comprendre leurs erreurs et répondez aux améliorations demandées.

Exercice 9 (★ Traduction en Langage C).

02_alice.c : Code d'Alice

```
import studio

def main :
    pi <- 3,14
    print(pi)
    # Pourquoi ça n'affiche pas pi ???
    exit()
```

Alice aimerait :

1. Régler le nombre de décimales après la virgule à deux décimales.
2. Finalement une manière automatique d'afficher le résultat proprement.
3. Et puis en notation scientifique, car c'est une vraie scientifique.

Exercice 10 (★★ Dépassement capacité d'un int).

03_bob.c : Code de Bob

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int gros_nombre;
    printf("Entrez un gros nombre : ");
    scanf("%d", gros_nombre);
    printf("%d, un gros nombre ?\n");
    gros_nombre = 999999999999999999;
    printf("%d, un gros nombre !\n");
    exit(EXIT_SUCCESS);
}
```

Bob aimerait :

1. Juste des gros nombres entier, les nombres négatifs ne l'intéressent pas, quel type lui permet d'avoir le plus gros nombre entier ?
2. Afficher le plus gros nombre possible avec ce type.
3. Prouver à Bob que c'est le plus gros nombre en l'affichant en hexadécimal.
4. Expliquer à Bob ce qui limite ses gros nombres avec le type `int`.

Exercice 11 (★★ Imprécision flottante).

04_charlie.c : Code de Charlie

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    float racine = 1.414213562373095048;
    printf("racine de 2 vaut %g\n", racine);
    printf("variable : %.18f\n", racine);
    printf("texte      : 1.414213562373095048\n");
    /* C'est différent, étrange ... */
    float clavier;
    printf("Recopiez :\n>1.414213562373095048\n>");
    scanf("%f", &clavier);
    printf("copie : %.18f\n", clavier);
    printf("texte : 1.414213562373095048\n");
    /* C'est la machine qui bug ! */
    exit(EXIT_SUCCESS);
}
```

Charlie aimerait :

1. Pouvoir écrire un nombre aussi précis que son texte dans son programme.
2. Que l'utilisateur puisse saisir un nombre aussi précis au clavier.

Exercice 12 (★★★ Partie entière d'un flottant).

Votre programme demande à l'utilisateur la saisie d'un réel. Le programme affiche ensuite à l'utilisateur la valeur flottante de ce réel et sa partie entière (chiffres avant la virgule). Ceci peut s'illustrer par les sorties suivantes :

```
Entrez un réel : 42.1337
La partie entière de 42.1337 est environ 42
```

```
Entrez un réel : 1e20
La partie entière de 1e+20 est environ 1000000000000000000000000000000000
```

```
Entrez un réel : 0.00001
La partie entière de 1e-05 est environ 0
```

```
Entrez un réel : 1e100
La partie entière de 1e+100 est environ 999999999999999999999999999999996693535
322073426194986990198284960792713915417520186694826443244189778401
17055488
```

Exercice 13 (★★★ Hexadécimal ?).

Oscar est passé en coup de vent nous donner du travail pour vous. Nous avons besoin de votre expertise pour écrire le programme suivant :

1. Sauvegarder fb40 8be9, mais pas sur un long, il dit que ça prend trop de mémoire pour si peu et l'afficher comme un entier positif.
2. Afficher l'hexadécimal (en majuscules) de 42 et '42' à côté.
3. Puis il a terminé sur une blague disant que vous qui êtes informaticien comprendriez, 'if 212 063 991 488 173 then 223 196 547 513 038', que veut-il dire par ces nombres ?

05_final.c : Code final

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    /* À vous de jouer : */

    exit(EXIT_SUCCESS);
}
```

Exercice 14 (★★★★ Un message secret).

Alice et Bob s'envoient des réels, mais Charlie ne comprend pas ce que peuvent signifier ces valeurs :

```
6.74219984e+22 1.77278228e+28 2.95631592e+21 7.71349945e+31
7.50357589e+28 4.54420104e+30 7.27183622e+22 2.72235332e+20
4.15838070e+21 7.03667266e+22 7.36963257e+28 9.20241483e-33
```

Oscar lui indique que Bob vient de découvrir que les entiers et flottants peuvent se modéliser sur une même plage de 4 octets d'après la norme IEEE 754 et qu'il sait qu'un caractère ASCII peut se sauvegarder sur 1 octet. Il a rigolé en disant que c'est facile avec le langage C et qu'il souhaite bonne chance aux développeurs Python.

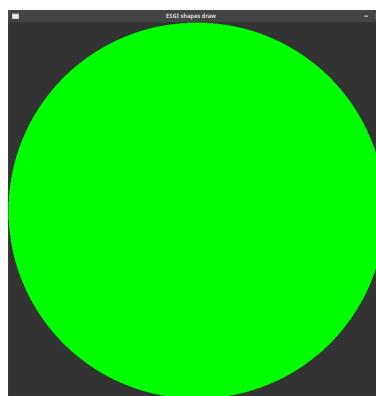
Exercice 15 (∞ Mini-projet : dessins de formes).

Charlie vous propose de réaliser un petit programme de dessin. Il a commencé la partie graphique et pense que vous pourriez l'aider à terminer son programme. Votre partie du travail serait la suivante :

- Lire un caractère dans `read_command` et le passer à `call_command`.
- Lire les informations d'un cercle (coordonnées du centre et rayon) puis sa couleur (intensités RGB) dans `read_circle` et les passer à `draw_circle`.
- Lire les informations d'une ligne (coordonnées du début et de la fin) puis sa couleur dans `read_line` et les passer à `draw_line`.

Par exemple le dessin d'un cercle vert de rayon 1 se ferait par l'interaction suivante :

```
Entrez une commande :  
- c : cercle (centre(x, y) rayon couleur(r, g, b))  
- l : ligne (debut(x, y) fin(x, y) couleur(r, g, b))  
- q : quitter  
>>> c 0 0 1 0 1 0  
Entrez une commande :  
- c : cercle (centre(x, y) rayon couleur(r, g, b))  
- l : ligne (debut(x, y) fin(x, y) couleur(r, g, b))  
- q : quitter  
>>> q
```



Un dessin plus avancé pourrait être donné par les commandes suivantes :

```
c 0 0 1 0 0 0
c 0 0 0.99 1 0.8 0
c 0 -0.3 0.4 1 1 1
c 0 0.2 0.7 1 0.8 0
l 0.3 -0.3 0.45 -0.35 0 0 0
l 0.45 -0.35 0.4 -0.5 0 0 0
c 0.4 0.25 0.25 1 1 1
c 0.4 0.25 0.2 0 0 0
c 0.4 0.25 0.19 0.2 0 0
c 0.4 0.25 0.1 0 0 0
c 0.3 0.35 0.05 1 1 1
c 0.5 0.15 0.025 1 1 1
c -0.4 0.25 0.25 1 1 1
c -0.4 0.25 0.2 0 0 0
c -0.4 0.25 0.19 0.2 0 0
c -0.4 0.25 0.1 0 0 0
c -0.5 0.35 0.05 1 1 1
c -0.3 0.15 0.025 1 1 1
```



À noter que cet exercice utilise la bibliothèque graphique SDL (se référer à la partie [V](#)). Ceci demande donc l'installation SDL 1.2 et l'ajout du linkage de la bibliothèque SDL dans la commande de compilation. La partie graphique a déjà été codée et vous est donnée par le code suivant. À noter que le copier-coller depuis le pdf peut introduire des symboles non compilables : une version .c est fournie dans l'archive associée à ce support. À vous de compléter les TODO pour faire fonctionner le code !

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI shapes draw";

int call_command(char shape_id);
/* Valeurs attendues entre -1 et 1 pour les coordonnées et 0 et 1 pour
   les couleurs : */
void draw_circle(double center_x, double center_y, double rayon, double
   r, double g, double b);
void draw_line(double start_x, double start_y, double end_x, double
   end_y, double r, double g, double b);

void read_circle() {
    /* TODO lire centre et rayon d'un cercle */
    /* TODO lire couleur RGB */
}

void read_line() {
    /* TODO lire début et fin d'une ligne */
    /* TODO lire couleur RGB */
}

int read_command() {
    /* TODO lire les commandes du joueur */
    return call_command('q');
}

int call_command(char shape_id) {
    switch(shape_id) {
        case 'c' : read_circle(); break;
        case 'l' : read_line(); break;
        case 'q' : return 0;
        default : break;
    }
    return 1;
}
```

```

SDL_Surface * ecran = NULL;

void draw_circle(double center_x, double center_y, double rayon, double
← r, double g, double b) {
    filledCircleRGBA(ecran, center_x * largeur / 2. + largeur / 2.,
    ← -center_y * hauteur / 2. + hauteur / 2.,
    rayon * fmin(largeur, hauteur) / 2., r * 255, g * 255, b
    ← * 255, 255);
}

void draw_line(double start_x, double start_y, double end_x, double
← end_y, double r, double g, double b) {
    lineRGBAlpha(ecran, start_x * largeur / 2. + largeur / 2., -start_y *
    ← hauteur / 2. + hauteur / 2.,
    end_x * largeur / 2. + largeur / 2., -end_y * hauteur /
    ← 2. + hauteur / 2., r * 255, g * 255, b * 255, 255);
}

int main(int argc, char * argv[]) {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
    ← SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
    SDL_Flip(ecran);

    while(read_command()) {

        SDL_Flip(ecran);
        SDL_Delay(1000 / 60);
    }

    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_SUCCESS);
}

```

La compilation de ce code peut se faire via les commandes suivantes :

- Pour Linux :

```
gcc -o prog 04_shapes.c -lSDL -lSDL_gfx -lm
```

- Pour Windows (via MSYS2 MinGW64) :

```
gcc -o prog 04_shapes.c -DSDL_MAIN_HANDLED -lmingw32
→ -lSDLmain -lSDL -lSDL_gfx -lm
```

(En cas de difficultés avec SDL 1.2, par exemple avec Mac, une alternative est donnée en SDL 2. Attention : ceci change `-lSDL -lSDL_gfx` en `-lSDL2 -lSDL2_gfx` dans la commande de compilation).

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL2_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI shapes draw";

int call_command(char shape_id);
/* Valeurs attendues entre -1 et 1 pour les coordonnées et 0 et 1 pour
→ les couleurs : */
void draw_circle(double center_x, double center_y, double rayon, double
→ r, double g, double b);
void draw_line(double start_x, double start_y, double end_x, double
→ end_y, double r, double g, double b);

void read_circle() {
    /* TODO lire centre et rayon d'un cercle */
    /* TODO lire couleur RGB */
}

void read_line() {
    /* TODO lire début et fin d'une ligne */
```

```

        /* TODO lire couleur RGB */
}

int read_command() {
    /* TODO lire les commandes du joueur */
    return call_command('q');
}

int call_command(char shape_id) {
    switch(shape_id) {
        case 'c' : read_circle(); break;
        case 'l' : read_line(); break;
        case 'q' : return 0;
        default : break;
    }
    return 1;
}

SDL_Renderer * ecran = NULL;

void draw_circle(double center_x, double center_y, double rayon, double
→ r, double g, double b) {
    filledCircleRGBA(ecran, center_x * largeur / 2. + largeur / 2.,
    → -center_y * hauteur / 2. + hauteur / 2.,
    → rayon * fmin(largeur, hauteur) / 2., r * 255, g * 255, b
    → * 255, 255);
}

void draw_line(double start_x, double start_y, double end_x, double
→ end_y, double r, double g, double b) {
    lineRGBA(ecran, start_x * largeur / 2. + largeur / 2., -start_y *
    → hauteur / 2. + hauteur / 2.,
    → end_x * largeur / 2. + largeur / 2., -end_y * hauteur /
    → 2. + hauteur / 2., r * 255, g * 255, b * 255, 255);
}

int main(int argc, char * argv[]) {
    SDL_Window * window = NULL;
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if(SDL_CreateWindowAndRenderer(largeur, hauteur, SDL_WINDOW_SHOWN,
    → &window, &ecran) != 0) {

```

```

fprintf(stderr, "Error in SDL_CreateWindowAndRenderer : %s\n",
        → SDL_GetError());
SDL_Quit();
exit(EXIT_FAILURE);
}
SDL_SetWindowTitle(window, titre);
SDL_SetRenderDrawColor(ecran, 51, 51, 51, 255);
SDL_RenderClear(ecran);
SDL_RenderPresent(ecran);

while(read_command()) {

    SDL_RenderPresent(ecran);
    SDL_Delay(1000 / 60);
}

SDL_DestroyRenderer(ecran);
SDL_DestroyWindow(window);
SDL_Quit();
exit(EXIT_SUCCESS);
}

```

Dans le cas où vous seriez amené à ajuster un code donné en SDL 1.2 en SDL 2, vous pourriez grossièrement commencer par modifier les lignes suivantes :

```

/* SDL 1.2 */
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

```

Modifier l'inclusion des entêtes :

```

/* SDL 2 */
#include <SDL2/SDL.h>
#include <SDL2/SDL2_gfxPrimitives.h>
/* SDL 2 : potentiellement sous Mac */
#include <SDL.h>
#include <SDL2_gfxPrimitives.h>

```

```

/* SDL 1.2 */
SDL_Surface * ecran = NULL;
if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
→ SDL_DOUBLEBUF)) == NULL) {

```

```

        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

```

Séparer sortie vidéo (fenêtre d'affichage) en une fenêtre et écran de rendu associé :

```

/* SDL 2 */
SDL_Window * window = NULL;
SDL_Renderer * ecran = NULL;
if(SDL_CreateWindowAndRenderer(largeur, hauteur, SDL_WINDOW_SHOWN,
    &window, &ecran) != 0) {
    fprintf(stderr, "Error in SDL_CreateWindowAndRenderer : %s\n",
    &SDL_GetError());
    SDL_Quit();
    exit(EXIT_FAILURE);
}
SDL_SetWindowTitle(window, titre);

```

```

/* SDL 1.2 */
SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
SDL_Flip(ecran);

```

Utiliser d'autres fonctions de remplissage de l'écran par une couleur unie et mettre à jour l'affichage pour l'utilisateur :

```

/* SDL 2 */
SDL_SetRenderDrawColor(ecran, 51, 51, 51, 255);
SDL_RenderClear(ecran);
SDL_RenderPresent(ecran);

```

3 Expressions

Souvent une variable sera une valeur qui sera amenée à changer : c'est-à-dire qu'elle peut être utile pour sauvegarder un résultat ou pour faire évoluer une donnée. Ceci peut être donné par un calcul et l'évaluation d'une expression. Une expression correspond par exemple à l'ensemble de nombres et d'opérations que l'on peut écrire sur une calculatrice pour obtenir un résultat.

3.1 Opérations classiques

3.1.1 Addition

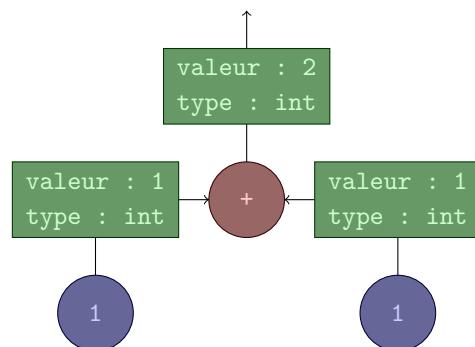
Une opération élémentaire est l'addition.

```
printf("1 + 1 = %d\n", 1 + 1);
```

En sortie :

```
1 + 1 = 2
```

Dans le code suivant, `1 + 1` est une expression que la machine évalue pour construire une valeur connue : `2` et peut ensuite l'afficher.

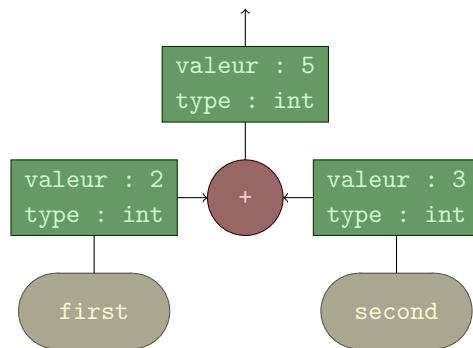


Dans une expression, lorsque le nom d'une variable est présent, celui-ci est remplacé par la valeur de cette variable puis l'expression est évaluée.

```
int first = 2;
int second = 3;
printf("%d + %d = %d\n", first, second, first + second);
```

En sortie :

```
2 + 3 = 5
```



À noter que le résultat de cette expression peut aussi être sauvegardé dans une variable en affectant à une variable le résultat de l'expression.

```
int first = 2;
int second = 3;
int resultat;
resultat = 2 + 3;
printf("%d + %d = %d\n", first, second, resultat);
```

En sortie :

```
2 + 3 = 5
```

3.1.2 Soustraction et multiplication

D'autres opérations classiques sont possibles sur les entiers comme la soustraction avec le tiret du milieu '-' ou la multiplication avec l'étoile '*' :

```
int first = 2;
int second = 3;
printf("%d - %d = %d\n", first, second, first - second);
printf("%d * %d = %d\n", first, second, first * second);
```

En sortie :

```
2 - 3 = -1
2 * 3 = 6
```

À noter que le langage C respecte la règle de priorité de la multiplication sur l'addition. Pour forcer une priorité, ceci se fait à l'aide de parenthèses.

```
printf("2 * 3 + 5 = %d\n", 2 * 3 + 5);
printf("2 * (3 + 5) = %d\n", 2 * (3 + 5));
```

En sortie :

```
2 * 3 + 5 = 11
2 * (3 + 5) = 16
```

Exercice 16 (★ Calculer pour l'utilisateur).

Écrire un code qui demande deux nombres réels à l'utilisateur, puis lui afficher :

1. L'addition de ces nombres.
2. La soustraction de ces nombres.
3. La multiplication de ces nombres.

3.1.3 Compatibilité entre entiers et flottants

À noter que ces opérations marchent aussi avec les nombres flottants et sont compatibles entre entiers et flottants. La subtilité à prendre en compte avec les nombres flottants est que du fait qu'ils représentent une approximation des nombres à virgule, ils peuvent introduire des imprécisions numériques :

```
float first = 4.9f;
float second = 4.3f;
int third = 1;
printf("%f + %f + %d = %f\n", first, second, third, first + second
→ + third);
```

En sortie :

```
4.900000 + 4.300000 + 1 = 10.200001
```

3.2 Division

3.2.1 Division entière

La division peut s'appliquer à l'aide du symbole slash '/'. Lorsqu'elle est appliquée à deux entiers, son résultat est celui de la division entière (le quotient de la division euclidienne) :

```
printf("4 / 2 = %d\n", 4 / 2);
printf("3 / 2 = %d\n", 3 / 2);
```

En sortie :

```
4 / 2 = 2
3 / 2 = 1
```

3.2.2 Modulo : reste de la division euclidienne

Comme pour la division euclidienne posée, il est possible de récupérer le reste à l'aide du symbole '%' : opération qui se nommera 'modulo' :

```
printf("7 = %d * 2 + %d\n", 7 / 2, 7 % 2);
```

En sortie :

```
7 = 3 * 2 + 1
```

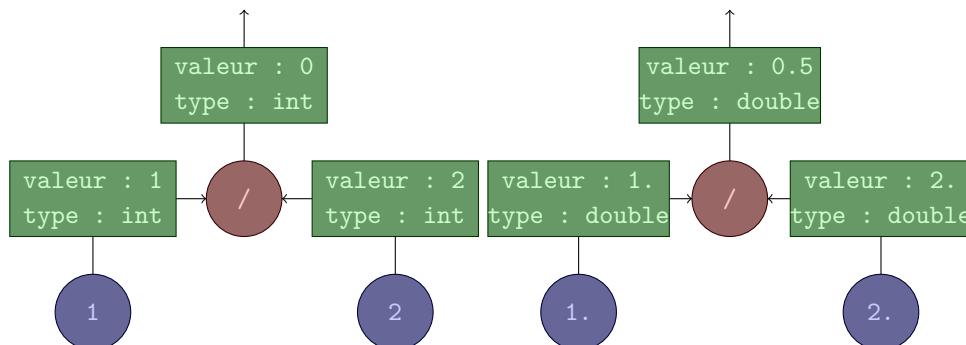
3.2.3 Division fractionnaire

Une remarque que l'on peut faire est que si on sauvegarde le résultat d'une division entière dans un flottant, ceci ne permet pas d'avoir un nombre à virgule. Cependant si l'un des deux nombres est flottant nous aurons l'évaluation de la division fractionnaire :

```
float resultat = 1 / 2;
printf("1 / 2 = %g\n", resultat);
printf("1. / 2 = %g\n", 1. / 2);
printf("1 / 2. = %g\n", 1 / 2.);
printf("1. / 2. = %g\n", 1. / 2.);
```

En sortie :

```
1 / 2 = 0
1. / 2 = 0.5
1 / 2. = 0.5
1. / 2. = 0.5
```



À noter qu'en l'absence de parenthèses, le choix de priorité d'évaluations de la division se fait de la gauche vers la droite.

```
printf("2. / 3. / 5. = %g\n", 2. / 3. / 5.);
printf("(2. / 3.) / 5. = %g\n", (2. / 3.) / 5.);
printf("2. / (3. / 5.) = %g\n", 2. / (3. / 5.));
```

En sortie :

```
2. / 3. / 5. = 0.133333
(2. / 3.) / 5. = 0.133333
2. / (3. / 5.) = 3.33333
```

En cas de doute sur l'ordre de priorité dans les évaluation, optez pour un parenthésage explicite.

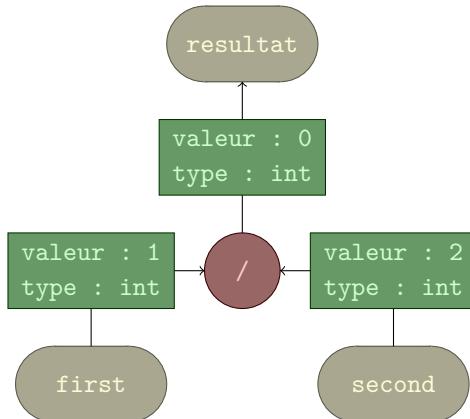
3.3 Coercition : un changement de type

Cependant, dans certains cas où l'on souhaite une division fractionnaire, il se peut que les entiers soient issus de variables :

```
int first = 1;
int second = 2;
float resultat;
resultat = first / second;
printf("%d / %d = %g\n", first, second, resultat);
```

En sortie :

```
1 / 2 = 0
```



Pour ceci, il est nécessaire de transformer au moins un entier en nombre flottant. Deux manières en utilisant les outils vus précédemment sont possibles :

1. (À éviter) Sauvegarder la valeur au préalable dans un flottant.

2. (Acceptable) Multiplier par 1. :

```
1 int first = 1;
2 int second = 2;
3 float resultat;
4 resultat = (first * 1.) / second;
5 printf("%d / %d = %g\n", first, second, resultat);
```

En sortie :

```
1 / 2 = 0.5
```

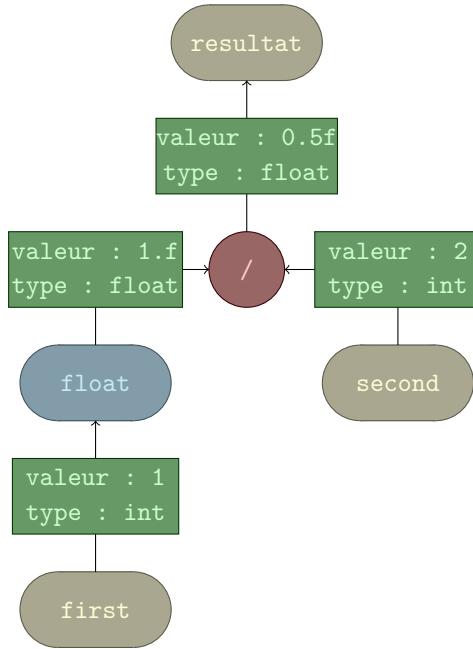
Une autre méthode est de forcer le changement de signe : ceci se fait en précisant le type à obtenir entre parenthèse avant l'expression lors de la conversion.

```
1 int first = 1;
2 int second = 2;
3 float resultat;
4 resultat = (float)first / second;
5 printf("%d / %d = %g\n", first, second, resultat);
```

En sortie :

```
1 / 2 = 0.5
```

Ceci, permet à la ligne 4 de passer de `int` à `float` pour `first` (la conversion de type est prioritaire). Ce concept est la coercition de type, souvent on l'entendra et on en parlera comme le fait de 'caster' un type.



Ce type d'opération est aussi utile pour utiliser un plus grand contenant dans le cas où l'opération dépasse ce que peut supporter le type, par exemple de `int` à `long` :

```

int first = 1000000;
int second = 1000000;
long resultat = first * second;
printf("(sans cast) %d * %d = %ld\n", first, second, resultat);
printf("(avec cast) %d * %d = %ld\n", first, second, (long)first *
      second);
  
```

En sortie :

```

(sans cast) 1000000 * 1000000 = -727379968
(avec cast) 1000000 * 1000000 = 1000000000000000
  
```

3.4 Réaffectation d'une variable avec opérateur

Nous avons vu que nous pouvons affecter une valeur à une variable depuis une expression. Ceci se fait souvent pour ajouter 1 à une variable.

```
int nombre = 1;
printf("nombre = %d\n", nombre);
nombre = nombre + 1;
printf("nombre = %d\n", nombre);
```

En sortie :

```
nombre = 1
nombre = 2
```

En réalité ce type d'écriture peut être abrégée : si on applique une opération à une variable, il est possible de le noter comme une affectation avec l'opérateur collé au symbole '=' :

```
int nombre = 1;
printf("nombre = %d\n", nombre);
nombre += 1;
printf("(+ 1) nombre = %d\n", nombre);
nombre *= 5;
printf("(* 5) nombre = %d\n", nombre);
nombre -= 3;
printf("(- 3) nombre = %d\n", nombre);
nombre %= 2;
printf("(%% 2) nombre = %d\n", nombre);
```

En sortie :

```
nombre = 1
(+ 1) nombre = 2
(* 5) nombre = 10
(- 3) nombre = 7
(%% 2) nombre = 1
```

Dans le cas où on ajoute 1 ou où on soustrait 1, ceci peut se simplifier par simplement l'opérateur unaire ++ ou -- (avant ou après le nom de la variable).

```
int i = 0;
i++; printf("i = %d\n", i);
++i; printf("i = %d\n", i);
i--; printf("i = %d\n", i);
--i; printf("i = %d\n", i);
```

En sortie :

```
i = 1
i = 2
i = 1
i = 0
```

Il faut noter que ces opérations qui changent la valeur de la mémoire sont dites avec 'effets de bord'. Mais tout comme les opérations classiques, elles renvoient une valeur :

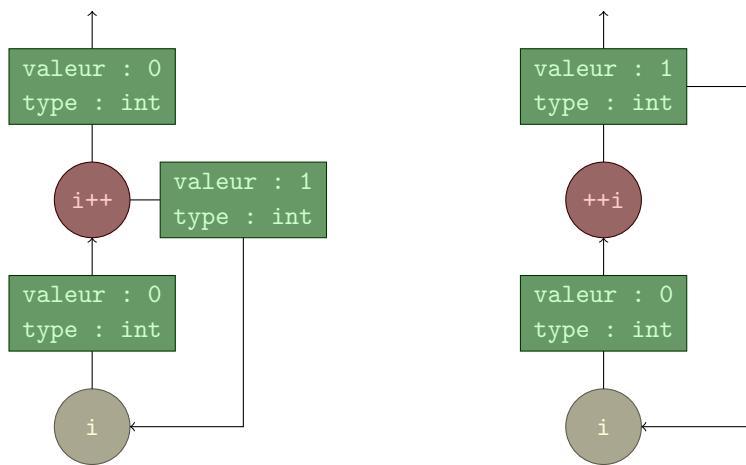
```
int a = 21;
int b = 3;
int c = 4;
printf("%d, %d, %d\n", a, b, c);
c = b = a *= 2; /*c = (b = (a *= 2))*/
printf("%d, %d, %d\n", a, b, c);
```

En sortie :

```
21, 3, 4
42, 42, 42
```

Dans le cas particulier de l'incrémentation `++` et la décrémentation `--`, la valeur de l'expression dépend du placement de l'opérateur :

- `i++` renvoie la valeur de `i` avant modification.
- `++i` renvoie la valeur de `i` après modification (comme les autres affectations et réaffectations).



```
int i = 1;
printf("i++ = %d\n", i++);
printf("++i = %d\n", ++i);
printf("i = %d\n", i);
```

En sortie :

```
i++ = 1
++i = 3
i = 3
```

3.5 Résumé

Opérateurs arithmétiques (entiers) :

```
int a, b;
a + b; // addition
a - b; // soustraction
a * b; // multiplication
a / b; // division entière
a % b; // modulo : reste de la division euclidienne
```

Opérateurs flottants :

```
float a, b;
a + b; // addition
a - b; // soustraction
a * b; // multiplication
a / b; // division fractionnaire
```

Affectation :

```
a = [VALEUR]; /* expression renvoie a */
b = a = [VALEUR];
```

Réaffectations avec opérateur :

```
a = a [OP] b; /* <=> */ a [OP]= b;
a = a + b; /* <=> */ a += b;
a = a - b; /* <=> */ a -= b;
a = a * b; /* <=> */ a *= b;
a = a / b; /* <=> */ a /= b;
a = a % b; /* <=> */ a %= b;
a = a + 1;
/* <=> */ a++; /* renvoie a avant modification */
/* <=> */ ++a; /* renvoie a après modification */
a = a - 1;
/* <=> */ a--; /* renvoie a avant modification */
/* <=> */ --a; /* renvoie a après modification */
```

Conversion de type :

```
(type)variable;
```

Résultat flottant pour division d'entiers :

```
int a, b;  
(a * 1.) / b;  
(float)a / b;
```

Dépassement de capacité d'un int :

```
int a, b;  
long res;  
res = (long)a * b;
```

3.6 Entrainement

Exercice 17 (★★ Opérations).

Le chat de votre binôme a renversé son café sur son ordinateur. Vous lui sauveriez la mise en terminant sa partie.

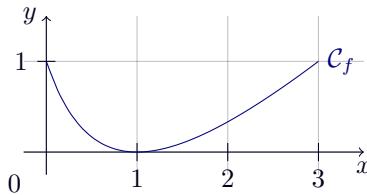
01_binome.c : Code de votre binôme

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int a, b;
    /* TODO : demander des valeurs pour a et b */
    /* TODO : afficher l'addition de a et b */
    /* TODO : échanger les valeurs de a et de b */
    /* TODO : afficher la soustraction de a et b */
    long c;
    /* TODO : affecter à c le résultat de la
       ↳ multiplication de a et b */
    float d;
    /* TODO : affecter à d le résultat de la division
       ↳ fractionnaire de a et b */
    exit(EXIT_SUCCESS);
}
```

Exercice 18 (★★ Division par zéro).

Alice est totalement perdue et a besoin de votre aide. Son programme lui dit '**Exception en point flottant (core dumped)**'. Elle veut juste calculer des valeurs de la fonction $x \mapsto \frac{(x-1)^2}{x+1}$. Elle a réussi à tracer la fonction sur Geogebra pour vérifier et vous donne son graphique. Aidez Alice.



02_alice.c : Code d'Alice

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int x = 0;
    int y = x - 1 * x - 1 / x + 1;
    printf("f(%d) = %d\n", x, y);
    x = 1;
    y = x - 1 * x - 1 / x + 1;
    printf("f(%d) = %d\n", x, y);
    x = 3;
    y = x - 1 * x - 1 / x + 1;
    printf("f(%d) = %d\n", x, y);
    exit(EXIT_SUCCESS);
}
```

Exercice 19 (★★ Affection d'une addition ?).

Bob est fier car son programme compile et fonctionne. Que pensez-vous de son code ? Pourriez-vous proposer à Bob une autre manière d'écrire son code pour plus de lisibilité et éviter des problèmes inattendus ensuite ?

03_bob.c : Code de Bob

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    long big = 0;
    int ajout;
    printf("big vaut %ld, faites le grossir : ", big);
    scanf("%d", &ajout);
    big = ajout+++big;
    printf("big vaut %ld !\n", big);
    exit(EXIT_SUCCESS);
}
```

Exercice 20 (★★ Imprécision et opérations).

Charlie a un problème, lorsqu'il ajoute un flottant pour avancer de pas en pas, ça ne change pas sa variable. Proposez à Charlie un changement pour faire fonctionner son code.

04_charlie.c : Code de Charlie

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    float resultat = 1.;
    float ajout = 1e-9;
    printf("resultat = %.15f\n", resultat);
    printf("on ajoute %.15f\n", ajout);
    resultat += ajout;
    printf("resultat = %.15f\n", resultat);
    /* Pourquoi resultat ne change pas ? */
    exit(EXIT_SUCCESS);
}
```

Exercice 21 (★★★ Message codé).

Oscar vous indique que $p = 4\ 285\ 404\ 239$ est un nombre sûr pour échanger des messages secrets sans être lu par vos camarades de classes. Il vous dit que vous pouvez lui envoyer et recevoir des messages de sa part avec la clé $k = 2\ 015\ 201\ 261$. Pour ça il faut juste multiplier le nombre à envoyer ou celui reçu par k puis calculer le reste de la division euclidienne par p . Pour tester, Oscar vous envoie le message '0x5c003212'. Écrivez un programme qui permet de lire ce message.

05_final.c : Code final

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    /* À vous de jouer : */

    exit(EXIT_SUCCESS);
}
```

Exercice 22 (★★ Afficher sinus et cosinus d'un angle).

Vous devez proposer un programme demandant à l'utilisateur la saisie d'un angle en degrés puis calculer sinus et cosinus de cet angle. Ceci demandera l'utilisation de l'entête `math.h` et la liaison de la bibliothèque `cmath` dans la commande de compilation par un `-lm`. Votre programme pourrait proposer les sorties suivantes :

```
Entrez un angle : 90
cos(90°) = 6.12323e-17
sin(90°) = 1
```

```
Entrez un angle : 45
cos(45°) = 0.707107
sin(45°) = 0.707107
```

```
Entrez un angle : 135
cos(135°) = -0.707107
sin(135°) = 0.707107
```

```
Entrez un angle : 20
cos(20°) = 0.939693
sin(20°) = 0.34202
```

Exercice 23 (★★★ Conversion d'une adresse IP).

Alice a découvert que sa machine était référencée par une adresse IP pour communiquer via le réseau internet. Bob lui explique qu'en réalité une adresse IP n'est rien de plus qu'un entier sur 4 octets. Alice souhaite donc faire la conversion d'un entier à une adresse IPv4 et réciproquement :

```
Entrez un entier : 4215377173
4215377173 = 251.65.141.21
Saisir une adresse IPv4 : 127.0.0.1
127.0.0.1 = 2130706433
```

Exercice 24 (★★★ Suite de Fibonacci par la formule de Binet).

Charlie a entendu parler de la suite de Fibonacci. A priori, cette suite pourrait se calculer par la formule mathématique suivante :

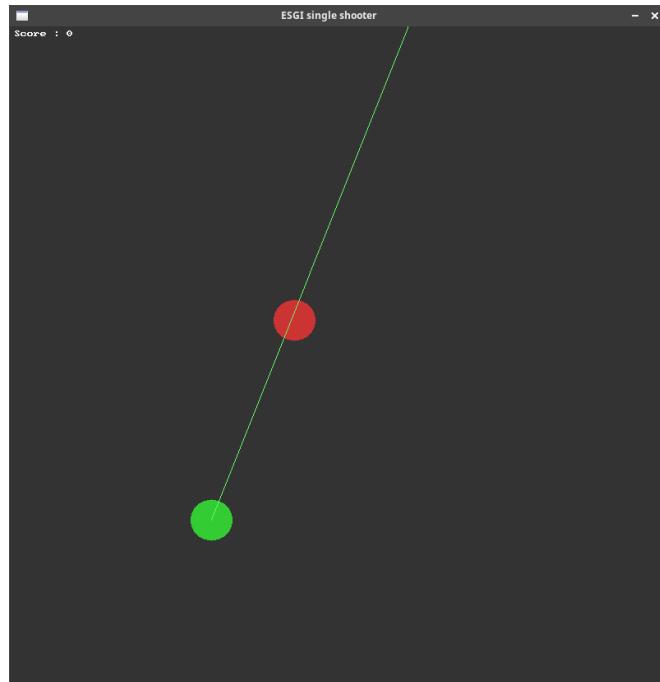
$$\mathcal{F}_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right)$$

Charlie vous informe qu'avec uniquement la base du Langage C, si c'est codé correctement, la méthode informatique est précise jusqu'à \mathcal{F}_{90} et qu'ensuite ceci resterait une approximation pertinente.

Exercice 25 (∞ Shooter).

Oscar est fatigué de coder dans un terminal et trouverait plus fun de commencer à créer des jeux sous interface graphique. Il a commencé à étudier la partie sur SDL dans le cours et vous a fait un squelette d'un mini-jeu de tir. Le but est de coder des fonctionnalités qui permettent à une personne de se déplacer et tirer sur un adversaire en mouvement.

Il aimeraient que le jeu ressemble au moins à l'image suivante :



Il vous a fait le code graphique en SDL 1.2 et aimeraient que vous complétriez les TODO avec des instructions. Il vous indique que pour que le jeu soit encore meilleur, il peut être intéressant de regarder la section suivante pour éviter que le joueur dépassé la souris.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI single shooter";

/* Coordonnées du joueur : */
int px, py;
/* Ligne de tir du joueur : */
int slpx, slpy;
/* Vitesse du joueur : */
int pv;

/* Coordonnées de l'adversaire : */
int ax, ay;
/* Vitesse de l'adversaire : */
int av;

void joueur_se_dirige_vers(int tx, int ty) {
    /* TODO : déplacer le joueur (px, py) vers la cible (tx, ty) */
    /* par pas de (pv) */
}

void adversaire_se_dirige_vers(int tx, int ty) {
    /* TODO : déplacer l'adversaire (ax, ay) vers la cible */
    /* (tx, ty) par pas de (av) */
}

void joueur REGARDER(int tx, int ty) {
    /* TODO : orienter la ligne de tir (slpx, slpy) vers */
    /* (tx, ty) */
}

int distance_tir(int x, int y) {
```

```

/* TODO : calculer la distance de la position (x, y) à la ligne de tir et remplacer 0x7fffffff par cette valeur */
return 0x7fffffff;
}

void place_adversaire() {
/* TODO : placer l'adversaire (ax, ay) aléatoirement en bordure de la fenêtre (largeur, hauteur) */
}

int distance_joueur(int x, int y) {
/* TODO : calculer la distance d'une position (x, y) au joueur (px, py) */
return 0x7fffffff;
}

void affichage(SDL_Surface * ecran) {
/* Remplissage de l'écran par un gris foncé uniforme : */
SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
/* Affichage du joueur : */
filledCircleRGBA(ecran, px, py, 25, 51, 204, 51, 255);
/* Affichage de l'adversaire : */
filledCircleRGBA(ecran, ax, ay, 25, 204, 51, 51, 255);

lineRGBA(ecran, px, py, slpx, slpy, 102, 255, 102, 255);
}

int main() {
srand(time(NULL));
/* Crédit à une fenêtre SDL : */
if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
    fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
    exit(EXIT_FAILURE);
}
SDL_Surface * ecran = NULL;
if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
    → SDL_DOUBLEBUF)) == NULL) {
    fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
    SDL_Quit();
    exit(EXIT_FAILURE);
}
SDL_WM_SetCaption(titre, NULL);

/* Placement du joueur au centre de la fenêtre : */
px = ecran->w / 2;

```

```
py = ecran->h / 2;
pv = 10;

ax = ecran->w / 2;
ay = 0;
av = 5;

int active = 1;
int grab = 0;
SDL_Event event;
int last_mouse_x = px;
int last_mouse_y = py;

int score = 0;
char display[100];

while(active) {

    affichage(ecran);
    sprintf(display, "Score : %d", score);
    stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
    SDL_Flip(ecran);

    while(SDL_PollEvent(&event)) {

        switch(event.type) {
            /* Utilisateur clique sur la croix de la fenêtre : */
            case SDL_QUIT : {
                active = 0;
            } break;

            /* Utilisateur enfonce une touche du clavier : */
            case SDL_KEYDOWN : {
                switch(event.key.keysym.sym) {
                    /* Touche Echap : */
                    case SDLK_ESCAPE : {
                        active = 0;
                    } break;
                }
            } break;

            /* Utilisateur commence le clic : */
            case SDL_MOUSEBUTTONDOWN : {
                switch(event.button.button) {
                    case SDL_BUTTON_LEFT : {
```

```

        grab = 1;
        last_mouse_x = event.button.x;
        last_mouse_y = event.button.y;
    } break;

    case SDL_BUTTON_RIGHT : {
        if(abs(distance_tir(ax, ay)) < 25) {
            ++score;
            ++pv;
            ++av;
            place_adversaire();
        }
    } break;
}

/* Utilisateur relâche le clic : */
case SDL_MOUSEBUTTONUP : {
    switch(event.button.button) {
        case SDL_BUTTON_LEFT : {
            grab = 0;
        } break;
    }
} break;

/* Utilisateur bouge la souris : */
case SDL_MOUSEMOTION : {
    last_mouse_x = event.motion.x;
    last_mouse_y = event.motion.y;
} break;
}

if(grab) {
    joueur_se_dirige_vers(last_mouse_x, last_mouse_y);
}
joueur_regarde(last_mouse_x, last_mouse_y);
adversaire_se_dirige_vers(px, py);
if(distance_joueur(ax, ay) < 25) {
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 204, 51, 51));
    filledCircleRGBA(ecran, px, py, 25, 0, 0, 0, 255);
    stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
    SDL_Flip(ecran);
    SDL_Delay(1000);
    active = 0;
}
}

```

```
    SDL_Delay(1000 / 60);
}

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

4 Conditions

4.1 Sélection d'instructions avec if

Selon la configuration des données, il est possible de vouloir choisir des traitements différents. Il est possible de se dire que si un test est vérifié, alors on applique un certain traitement.

Par exemple, pour éviter d'être à découvert, nous pouvons vérifier que nous avons l'argent pour acheter le nouveau jeu vidéo à la mode :

```
const int prix_jeu = 60;
int argent = 0;
printf("Combien avez-vous d'argent ? ");
scanf("%d", &argent);

if(prix_jeu < argent) {
    printf("J'achète le jeu !\n");
} else {
    printf("Il faudra encore économiser ... \n");
}
```

Exemple de sortie 1 :

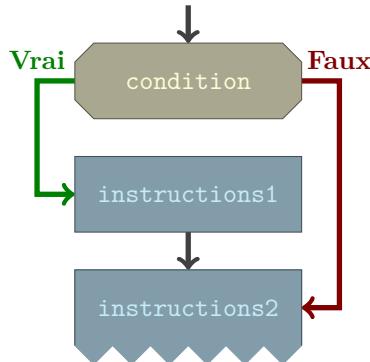
```
Combien avez-vous d'argent ? 42
Il faudra encore économiser ...
```

Exemple de sortie 2 :

```
Combien avez-vous d'argent ? 67
J'achète le jeu !
```

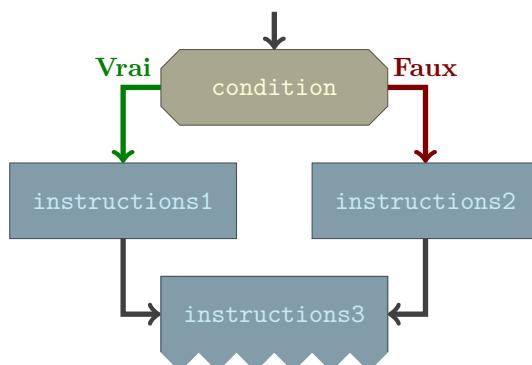
Ici, la syntaxe minimale attendue est celle du `if` auquel il faut fournir une condition de validation. Si cette condition est validée, alors ce qui suit le `if` sera exécuté. Pour cela on placera des accolades `{}` après le `if` dont le contenu sera lu si la condition est vérifiée :

```
if(condition) {
    /* instructions1 à lire si la condition est vraie. */
}
/* instructions2 */
```



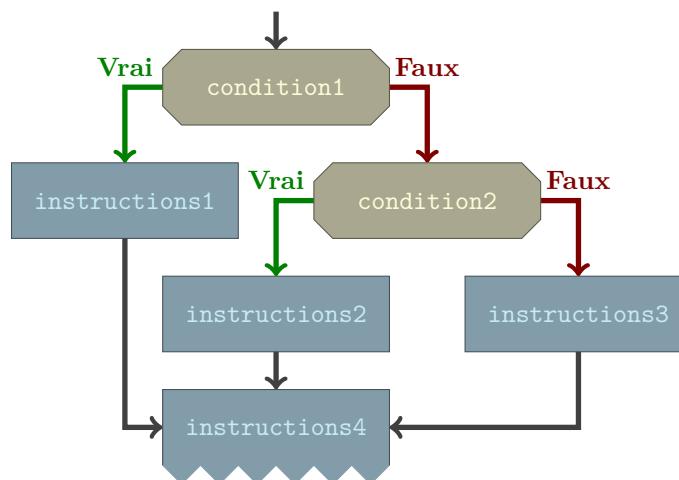
Il est aussi possible de préciser un comportement dans le cas où le test de la condition échoue. Pour cela on placera le mot clé **else** après le bloc d'instruction du **if** :

```
if(condition) {
    /* instructions1 à lire si la condition est vraie. */
} else {
    /* instructions2 à lire si la condition est fausse. */
}
/* instructions3 */
```



Il est aussi possible de chaîner des tests à réaliser au cas le précédent échoue en choisissant de vérifier une nouvelle condition par un `else if` :

```
if(condition1) {
    /* instructions1 à lire si la condition 1 est vraie. */
} else if(condition2) {
    /* instructions2 à lire si la condition 2 est vraie. */
} else {
    /* instructions3 à lire les conditions 1 et 2 sont fausses. */
}
/* instructions4 */
```



Ceci permet par exemple d'avoir une alternative dans notre exemple d'achat de jeu :

```
const int prix_jeu = 60;
const int prix_minecraft = 20;

int argent = 0;
printf("Combien avez-vous d'argent ? ");
scanf("%d", &argent);

if(prix_jeu < argent) {
    printf("J'achète le jeu !\n");
} else if(prix_minecraft < argent) {
```

```

    printf("J'achète minecraft !\n");
} else {
    printf("Il faudra encore économiser ... \n");
}

```

En sortie :

```

Combien avez-vous d'argent ? 42
J'achète minecraft !

```

4.2 Comparaisons

Les comparaisons sont un autre type d'opérateur qui regarde deux valeurs et renvoie une valeur de vérité si la condition est vérifiée : la valeur retournée sera 0 si c'est faux et une autre valeur si c'est vrai. Cette valeur de vérité peut ensuite être utilisée par notre instruction de contrôle `if` comme une condition.

Un premier opérateur de comparaison est par exemple `<` qui permet de vérifier que le nombre de gauche est strictement plus petit que le nombre de droite :

```

const int petit = 1;
const int grand = 42;
printf("petit < grand = %d\n", petit < grand);
printf("grand < petit = %d\n", grand < petit);
if(petit < grand) {
    printf("petit < grand !\n");
}
if(grand < petit) {
    printf("grand < petit ???\n");
}

```

En sortie :

```

petit < grand = 1
grand < petit = 0
petit < grand !

```

Notons que l'opérateur `=` effectue l'affectation d'une valeur dans une variable. Pour tester l'égalité, l'opérateur de comparaison est `==`

```
const int deux = 2;
printf("2 == deux ? valeur de vérité : %d\n", 2 == deux);
printf("1 == deux ? valeur de vérité : %d\n", 1 == deux);
```

En sortie :

```
2 == deux ? valeur de vérité : 1
1 == deux ? valeur de vérité : 0
```

Une liste plus exhaustive d'opérateurs de comparaison est la suivante :

Comparaison	Correspondance
a == b	Égalité : vrai si a est la même valeur que b
a != b	Différence : vrai si a est une valeur différente de b
a < b	vrai si a est strictement plus petit que b
a <= b	vrai si a est plus petit ou égal à b
a > b	vrai si a est strictement plus grand que b
a >= b	vrai si a est plus grand ou égal à b

À noter qu'une valeur de vérité peut être sauvegardée dans un variable et passée à une condition :

```
const int deux = 2;
int verification;
verification = deux == 2;
printf("verification = %d\n", verification);
if(verification) {
    printf("deux == 2\n");
} else {
    printf("deux != 2\n");
}
```

En sortie :

```
verification = 1
deux == 2
```

Exercice 26 (★ Déterminer une catégorie d'âge).

Écrire un code qui demande son âge à l'utilisateur puis lui affiche la catégorie des groupes établis selon le cycle de vie correspondante :

- **Enfant** : moins de 14 ans.
- **Adolescent** : de 15 à 24 ans.
- **Adulte** : de 25 à 64 ans.
- **Aîné** : plus de 65 ans.

4.3 Opérateurs booléens

Il est possible de construire des expressions booléennes en assemblant des valeurs de vérités par des opérateurs booléens. Ceci est utile par exemple pour vérifier deux conditions en même temps.

4.3.1 Intersection

On peut vouloir vérifier que plusieurs conditions sont vraies en même temps. Par exemple si l'on souhaite vérifier qu'un point (x, y) appartient à une boîte de sommets $(0, 0)$ et $(1, 1)$. Pour faire ceci nous allons utiliser l'opérateur d'intersection 'et' donné en langage C par **&&** :

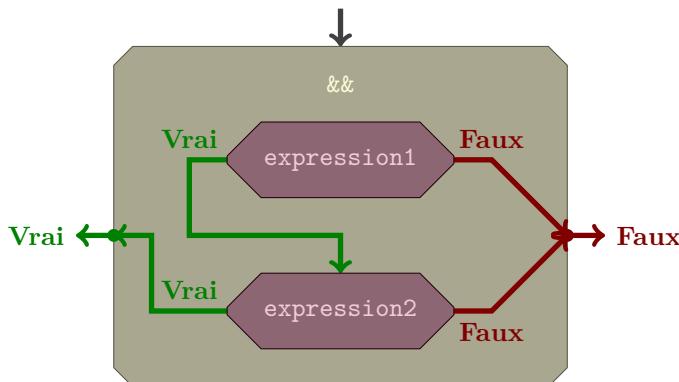
```
float x, y;
printf("Entrez des coordonnées x y : ");
scanf("%f %f", &x, &y);
if((x >= 0) && (x <= 1) && (y >= 0) && (y <= 1)) {
    printf("Bien joué : (%g, %g) est dans la boîte ((0, 0), (1,
    ↪ 1))\n", x, y);
} else {
    printf("Rate : (%g, %g) n'est pas dans la boîte ((0, 0), (1,
    ↪ 1))\n", x, y);
}
```

En sortie :

```
Entrez des coordonnées x y : 0.42 0.37
Bien joué : (0.42, 0.37) est dans la boîte ((0, 0), (1, 1))
```

En sortie :

```
Entrez des coordonnées x y : 1.01 0.5
Raté : (1.01, 0.5) n'est pas dans la boîte ((0, 0), (1, 1))
```



En effet, si une des conditions n'est pas vérifiée, alors le résultat est faux. Il est possible de l'observer en construisant la table de vérité l'opérateur d'intersection :

```
printf("&& | 0 | 1 |\n");
printf("----+---+---+\n");
printf(" 0 | 0 : 0 |\n";
printf("----+ - + - +
 1 | 0 : 1 |
----+---+---+
```

En sortie :

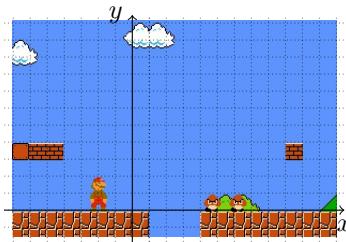
```
&& | 0 | 1 |
----+---+---+
 0 | 0 : 0 |
----+ - + - +
 1 | 0 : 1 |
----+---+---+
```

Exercice 27 (★ Intersection d'événements).

Écrire un code qui demande son âge à l'utilisateur puis lui indiquer s'il est dans la catégorie des jeunes de 18 à 25 ans.

4.3.2 Union

Un autre opérateur est l'opérateur de réunion 'ou inclusif' donné en langage C par `||`. Cet opérateur permet de valider un test si au moins une des valeurs est vraies.

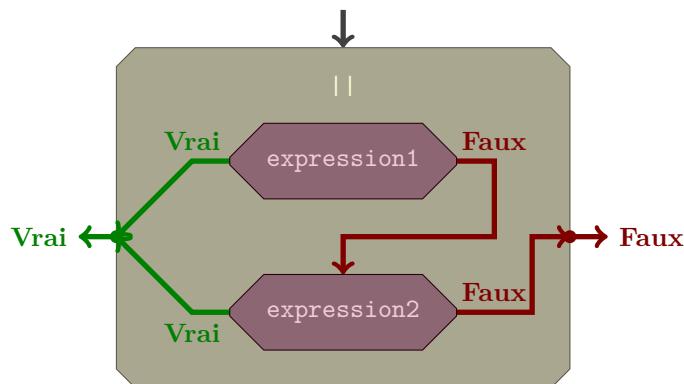


Par exemple, sur l'illustration ci-contre du jeu Super Mario Bros., la plateforme est séparée en deux morceaux. Nous pourrions donc vouloir vérifier que le personnage est sur l'une des plateformes à l'aide de son abscisse x . En effet, si $x \leq 1$ le personnage est sur le sol, mais si $x \geq 4$ est aussi sur le sol. Cette situation peut se traduire par le code suivant :

```
float x_personnage;
printf("Quelle est l'abscisse du personnage ? ");
scanf("%f", &x_personnage);
if((x_personnage <= 1) || (x_personnage >= 4)) {
    printf("Le personnage est sur le sol.\n");
} else {
    printf("Oups, regardez sous vos pieds !\n");
}
```

En sortie :

```
Quelle est l'abscisse du personnage ? -2
Le personnage est sur le sol.
```



De même que pour l'opérateur d'intersection, nous pouvons regarder la table de vérité de l'opérateur de réunion :

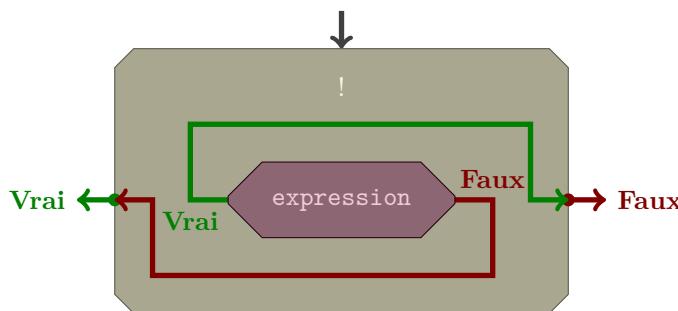
```
printf("|| | 0 | 1 |\n");
printf("----+---+\n");
printf(" 0 | %d : %d |\n", 0 || 0, 0 || 1);
printf("----+ - + - +\n");
printf(" 1 | %d : %d |\n", 1 || 0, 1 || 1);
printf("----+---+\n");
```

En sortie :

```
|| | 0 | 1 |
----+---+
 0 | 0 : 1 |
----+ - + - +
 1 | 1 : 1 |
----+---+
```

4.3.3 Négation

Un autre opérateur booléen unaire est celui de la négation. Il permet de changer la valeur de vérité d'un résultat et se place en C comme un `!` avant l'expression à évaluer.



Par exemple, on peut avoir une variable qui sauvegarde si un personnage peut lancer un sort et si le sort n'est plus disponible alors il ne peut pas être lancé :

```
int sort_disponible = 1;
if(! sort_disponible) {
```

```

    printf("Ce sort ne peut pas être lancé.\n");
} else {
    printf("C'est abracadabran !\n");
    sort_disponible = 0;
    printf("Le sort est maintenant épuisé.\n");
}

```

En sortie :

```
C'est abracadabran !
Le sort est maintenant épuisé.
```

De même une utilisation de l'opérateur de négation peut être utilisée comme expression pour maintenir l'état d'activité d'une case cochée, puis décochée, puis cochée :

```

int case_cochee = 1;
printf("case_cochée ? %d\n", case_cochee);
case_cochee = ! case_cochee;
printf("case_cochée ? %d\n", case_cochee);
case_cochee = ! case_cochee;
printf("case_cochée ? %d\n", case_cochee);

```

En sortie :

```
case_cochée ? 1
case_cochée ? 0
case_cochée ? 1
```

Exercice 28 (★★ Calculer l'amende d'un excès de vitesse).

Un étudiant révise le code de la route et s'interroge sur les excès de vitesse. Proposer un code qui demande vitesse et limitation puis calcule si l'utilisateur risque une amende et un retrait de points :

- Excès de vitesse inférieur à 20 km/h (avec vitesse maximale autorisée supérieure à 50 km/h) :
 - Amende forfaitaire de 68 euros ;
 - Retrait d'1 point sur permis de conduire.

- Excès de vitesse inférieur à 20 km/h (avec vitesse maximale autorisée inférieure ou égale à 50 km/h) :
 - Amende forfaitaire de 135 euros ;
 - Retrait d'1 point sur permis de conduire.
- Excès de vitesse égal ou supérieur à 20 km/h et inférieur à 30 km/h :
 - Amende forfaitaire de 135 euros ;
 - Retrait de 2 points sur permis de conduire.
- Excès de vitesse égal ou supérieur à 30 km/h et inférieur à 40 km/h :
 - Amende forfaitaire de 135 euros ;
 - Retrait de 3 points sur permis de conduire.
- Excès de vitesse égal ou supérieur à 40 km/h et inférieur à 50 km/h :
 - Amende forfaitaire de 135 euros ;
 - Retrait de 4 points sur permis de conduire.
- Excès de vitesse supérieur ou égal à 50 km/h :
 - Amende de 1 500 euros ;
 - Retrait de 6 points sur permis de conduire.

4.4 Ternaire : expression sous condition

Si nous devions naturellement donner le minimum de deux valeurs, nous utiliserions une comparaison pour tester l'état des variables, puis la structure **if-else** pour sauvegarder la plus petit valeur comme il suit :

```
int a, b;
int minimum;
printf("Entrez deux entiers : ");
scanf("%d %d", &a, &b);
if(a < b) {
    minimum = a;
} else {
    minimum = b;
}
printf("Le minimum de %d et %d est %d\n", a, b, minimum);
```

En sortie :

```
Entrez deux entiers : 42 1337
Le minimum de 42 et 1337 est 42
```

En langage C, il existe une syntaxe nommée ternaire qui reproduit ce schéma de **if-else** et se comporte comme une expression :

```
[CONDITION] ? [VALEUR SI VRAI] : [VALEUR SI FAUX]
```

Ceci permet d'écrire le code suivant qui se comporte comme le précédent :

```
int a, b;
int minimum;
printf("Entrez deux entiers : ");
scanf("%d %d", &a, &b);
minimum = (a < b) ? a : b;
printf("Le minimum de %d et %d est %d\n", a, b, minimum);
```

En sortie :

```
Entrez deux entiers : 42 1337
Le minimum de 42 et 1337 est 42
```

4.5 Switch : se brancher un à résultat

Un autre outil intéressant est le **switch**. Son intérêt est de pouvoir remplacer la **if-else** dans le cas où on regarde une succession d'égalités comme pour un menu.

```
printf("1 - Jouer\n2 - Options\n3 - Ragequit\n---\nVotre choix ?
→ ");
scanf("%d", &choix);
if(choix == 1) {
    printf("Que le jeu commence !\n");
} else if(choix == 2) {
    printf("Paramétrons ça.\n");
} else if(choix == 3) {
    printf("Bien, au revoir !\n");
} else {
```

```

printf("Hum, le choix %d, je ne comprends pas ce choix...\n",
      choix);
}

```

En sortie :

```

1 - Jouer
2 - Options
3 - Ragequit
---
Votre choix ? 1
Que le jeu commence !

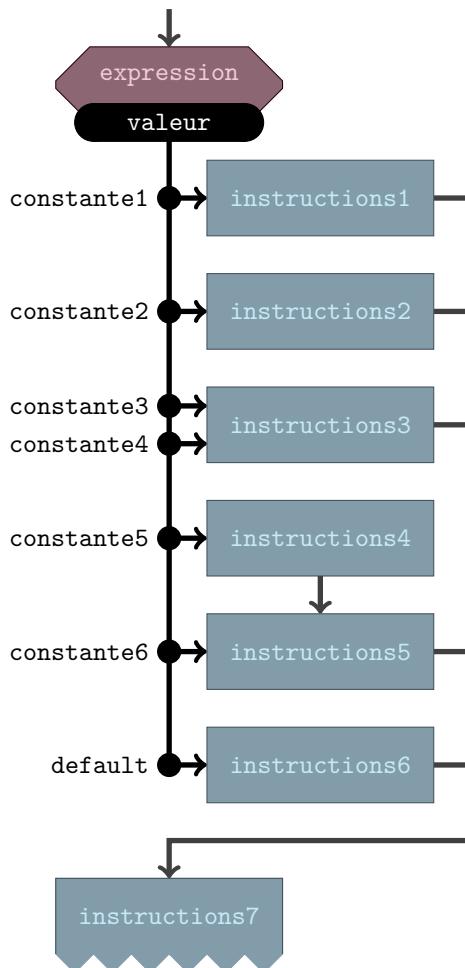
```

La syntaxe du **switch** peut se substituer à ce code : elle regarde une expression et saute à le label de la constante correspondant au résultat. À noter que pour sortir du **switch**, nous utiliserons l'instruction **break**; sinon le programme lit le code au label suivant :

```

switch(expression) {
    case [CONSTANTE 1] :
        /* si expression == [CONSTANTE 1] */
        break;
    case [CONSTANTE 2] :
        /* si expression == [CONSTANTE 2] */
        break;
    case [CONSTANTE 3] :
    case [CONSTANTE 4] :
        /* si expression == [CONSTANTE 3] ou expression == [CONSTANTE
           → 4] */
        break;
    case [CONSTANTE 5] :
        /* si expression == [CONSTANTE 5] */
    case [CONSTANTE 6] :
        /* si expression == [CONSTANTE 5] ou [EXPRESSION] ==
           → [CONSTANTE 6] */
        break;
    /* ... */
    default :
        /* équivalent du else */
}

```



Dans le cas de notre exemple de menu, ceci donnerait le code suivant :

```

int choix;
printf("1 - Jouer\n2 - Options\n3 - Ragequit\n---\nVotre choix ?
→ ");
scanf("%d", &choix);
switch(choix) {
    case 1 :
        printf("Que le jeu commence !\n");
        break;
    case 2 :
  
```

```
printf("Paramétrons ça.\n");
break;
case 3 :
    printf("Bien, au revoir !\n");
    break;
default :
    printf("Hum, le choix %d, je ne comprends pas ce choix... \n",
           choix);
}
```

En sortie :

```
1 - Jouer
2 - Options
3 - Ragequit
---
Votre choix ? 3
Bien, au revoir !
```

4.6 Résumé

Structure de contrôle sous condition :

```
if(condition1) {
    /* Instructions si condition1 vraie. */
} else if(condition2) {
    /* Instructions si condition1 fausse mais condition2 vraie. */
} else {
    /* Instructions si aucune des conditions n'est vraie. */
}
```

Tests de comparaison de nombres :

```
a == b; /* a est égal à b ? */
a != b; /* a est différent de b ? */
a < b; /* a est strictement plus petit que b ? */
a <= b; /* a est plus petit ou égal b ? */
a > b; /* a est strictement plus grand que b ? */
a >= b; /* a est plus grand ou égal b ? */
```

Opérateurs booléens :

```
a && b; /* ET : a et b sont vraies ? */
a || b; /* OU : a ou b est vraie ? */
!a; /* NON : a n'est pas vraie ? */
```

Expression ternaire :

```
[CONDITION] ? [VALEUR SI VRAIE] : [VALEUR SI FAUSSE];
```

Switch :

```
switch([EXPRESSION]) {
    case [CONSTANTE 1] :
        /* Instructions si [EXPRESSION] == [CONSTANTE 1] */
        break;
    /* ... */
    default :
        /* Instructions par défaut */
}
```

4.7 Entraînement

Exercice 29 (** Acheter un article).

Alice fait les soldes mais les vendeurs ont oublié d'afficher le prix soldé. Elle aimerait faire un programme pour calculer automatiquement si elle peut acheter un article. Pour ceci, elle fournit, pour chaque article, les informations suivantes :

1. Son argent.
2. Le prix de l'article hors soldes.
3. Le taux de remise en pourcentage.

Ceci pourrait donner la sortie suivante :

```
Votre argent : 42
Le prix de l'article (hors soldes) : 50
Remise en % : -30
L'article en solde vaut 35
J'achète !
```

Exercice 30 (★★ Menu et addition).

Alice souhaite ouvrir un fast-food pour que des étudiants puissent y manger rapidement sur le temps de midi. Pour faciliter les commandes, elles souhaite que les clients utilisent un automate. Elle fait donc appel à vous pour créer le programme de son automate. L'interaction avec l'automate pourrait donner la sortie suivante :

```
Voici nos entrées :  
1. (1.50 €) Salade racontée.  
2. (2.50 €) Oeufs embrouillés.  
0. Non merci.  
Votre choix : 1  
Voici nos plats :  
1. (12 €) Pizza spéciale informaticien.  
2. (8 €) Pâtes spéciales étudiants.  
0. Non merci.  
Votre choix : 2  
Voici nos desserts :  
1. (2.50 €) La cerise sur le gâteau.  
0. Non merci.  
Votre choix : 0  
Votre commande :  
- (1.50 €) Salade racontée.  
- (8 €) Pâtes spéciales étudiants.  
Total hors taxes : 9.50 €  
Total TTC 11.40 € (TVA à 20%)
```

Exercice 31 (★★★ Calculatrice).

Votre binôme vous laisse faire le menu, il complétera le reste après votre passage.

Le menu doit proposer les options suivantes :

1. Calculer.
2. Quitter.

Finalement, votre binôme n'a pas le temps de faire la partie calculer, il a piscine. Si l'utilisateur choisi 'Calculer', vous devez lui proposer d'entrer des entiers et une opération sous la forme '[NOMBRE 1] [OPERATEUR] [NOMBRE 2]' et calculer. [OPERATEUR] peut être l'addition, la soustraction, la multiplication, la division ou le modulo. Ce qui donne quelque chose comme la sortie suivante :

```
1 - Calculer
2 - Quitter
---
Votre choix : 1
>>> 6 * 7
6 * 7 = 42
```

Exercice 32 (★★ Racines d'un polynôme du second degré).

Bob aimerait automatiser la récupération des racines réelles d'un polynôme du second degré $ax^2 + bx + c$ pour faire un jeu avec des lancers de projectiles depuis son cours de mathématiques du lycée. On rappelle que :

- Si $a \neq 0$, on regarde le discriminant $\Delta = b^2 - 4ac$.
 - Si $\Delta < 0$, il n'y a pas de solution réelle.
 - Si $\Delta = 0$, $x_0 = \frac{-b}{2a}$ est l'unique solution.
 - Sinon, $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$ sont les deux solutions.
- Sinon si $b \neq 0$, il y a une solution $x_0 = \frac{-c}{b}$.
- Sinon c'est une fonction constante.

Proposez un code à Bob pour répondre à sa problématique.

Notez que $\sqrt{\cdot}$ se trouve comme `sqrt` dans la bibliothèque `math.h`. Dans la commande de compilation `gcc`, il faudra ajouter `-lm` pour l'utiliser.

Exercice 33 (★★★ Code obscurantiste).

Charlie se prépare pour l'**IOCCC**, une grande douleur pour son binôme qui essaie de le lire et ne comprend pas son code. Étudiez le concept proposé par Charlie, puis proposez un code tel que vous l'auriez écrit pour qu'il soit lisible d'un camarade.

04_charlie.c : Code de Charlie

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int a, b;
    printf("Entrez deux nombres : ");
    scanf("%d %d", &a, &b);
    if(!(a - b))
        printf("%d et %d sont égaux\n", a, b);
    else
        printf("Entre %d et %d, le plus petit est %d\n",
            ↪ a, b, ((long)(unsigned int)(a - b) == a - b) ?
            ↪ b : a);
    exit(EXIT_SUCCESS);
}
```

Exercice 34 (★★★ Validation d'un mot de passe).

Robert veut faire un système avec deux authentifications. L'utilisateur doit fournir les codes secrets '42' et '1337' pour avoir accès aux fonctionnalité de son logiciel. Il se réserve aussi un mot de passe administrateur '1235' qu'il faut écrire comme les deux codes secrets. Les combinaisons suivantes donnent accès au programme :

- 42 1337
- 1337 42
- 1235 1235

Robert ne souhaite pas que vous touchiez au code où il n'a pas placé de commentaires. Complétez son code pour l'aider à mettre en place ce système.

05_robert.c : Code de Robert

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int first_pass = 0, second_pass = 0;

    if(first_pass > second_pass) {
        /* Echanger first_pass et second_pass */

    }
    if(/* Condition : si mot de passe erroné */

    ) {
        printf("Accès refusé.\n");
        exit(EXIT_SUCCESS);
    }
    printf("Bienvenue !\n");
    exit(EXIT_SUCCESS);
}
```

Exercice 35 (★★★ Calcul de l'impôt sur le revenu).

Bob a enfin trouvé un travail dans l'informatique et commence à gagner de l'argent ! Il a cependant découvert qu'une partie de cet argent doit être reversée aux impôts. Il ne comprend rien à ce système de calcul par tranches et a besoin de votre aide. Charlie explique que le calcul par tranches c'est simple. Par exemple pour les tranches :

- Première tranche à 10% jusqu'à 10 000 €.
- Second tranche à 25% de 10 000 € à 20 000 €.
- Troisième tranche à 50% de 20 000 € à 50 000 €.
- Dernière tranche à 100% à partir de 50 000 €.

Des exemples de calculs pourraient être les suivants :

0	10 000	20 000	50 000
10%	25%	50%	100%

— Calcul de tranches sur 5 000 :

0	5 000	10 000	20 000	50 000
10%		25%	50%	100%
$\frac{10}{100} \times 5\ 000$				
500				

— Total : 500

— Calcul de tranches sur 15 000 :

0	10 000	15 000	20 000	50 000
10%		25%	50%	100%
$\frac{10}{100} \times 10\ 000$	$\frac{25}{100} \times 5\ 000$			
1 000	1 250			

— Total : 2 250

Calcul de tranches sur 40 000 :

0	10 000	20 000	40 000	50 000
	10%	25%	50%	
$\frac{10}{100} \times 10\ 000$	$\frac{25}{100} \times 10\ 000$	$\frac{50}{100} \times 20\ 000$		
1 000	2 500	10 000		

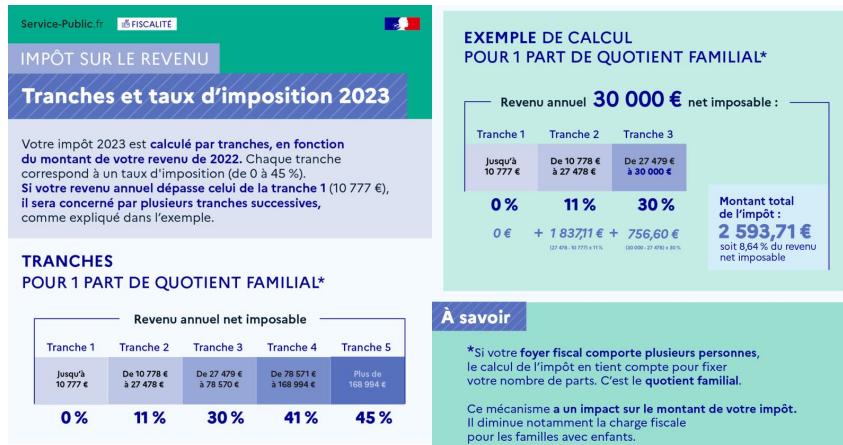
Total : 13 500

Calcul de tranches sur 70 000 :

0	10 000	20 000	50 000	70 000
	10%	25%	50%	100%
$\frac{10}{100} \times 10\ 000$	$\frac{25}{100} \times 10\ 000$	$\frac{50}{100} \times 30\ 000$	$\frac{100}{100} \times 20\ 000$	
1 000	2 500	15 000	20 000	

Total : 38 500

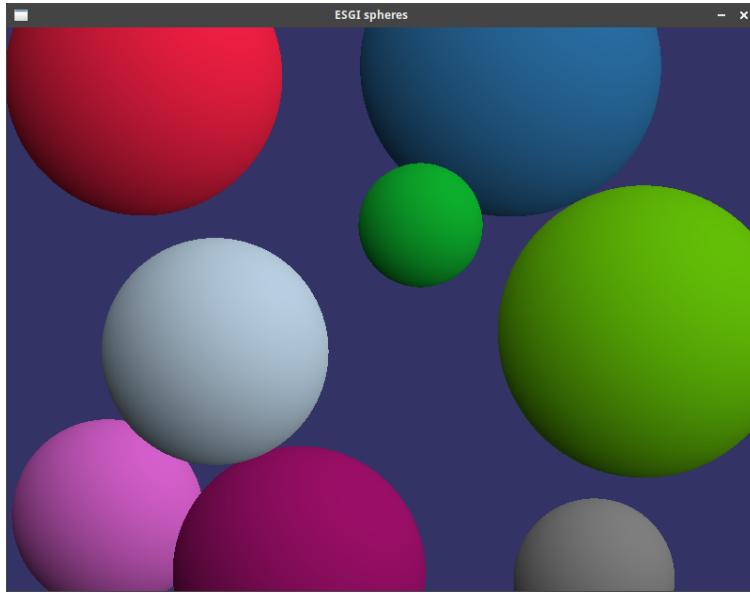
Bob vient de voir les informations officielles données sur le [site du gouvernement](#) et vous serait reconnaissant si vous pouviez réaliser un programme qui implémente ces concepts en pratique.



(Infographie issue de <https://www.service-public.fr/particuliers/vosdroits/F1419>).

Exercice 36 (∞ Lancé de rayon sur sphères).

Charlie aimerait réussir à faire un lancé de rayon sur des sphères :



Il est un peu perdu, mais Oscar lui donne quelques indications. Nous pouvons définir un rayon depuis une position initiale S et une direction \vec{v} depuis un paramètre réel t . C'est-à-dire que pour tout point Q sur la trajectoire définie par le rayon, il existe t tel que $Q = S + t\vec{v}$. Pour une sphère (ou cercle) définie par un centre C et un rayon r , nous avons que tout point Q appartenant à la sphère est tel que $\|Q - C\| = r$. Ceci revient donc à résoudre pour t l'équation suivante :

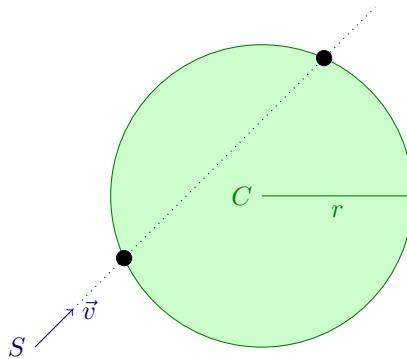
$$\|S + t\vec{v} - C\|_2^2 = r^2$$

Oscar dit que c'est trivial en utilisant les propriétés du produit scalaire pour cette norme euclidienne et la résolution d'un polynôme du second degré. Il a parlé de théorème de Pythagore, linéarité du produit scalaire, a gribouillé les équations suivante et est parti :

$$\|A\|^2 = \langle A \cdot A \rangle$$

$$\langle (A + \alpha B) \cdot C \rangle = \langle A \cdot C \rangle + \alpha \langle B \cdot C \rangle$$

$$\langle A \cdot B \rangle = A_x \times B_x + A_y \times B_y + A_z \times B_z = \langle B \cdot A \rangle$$

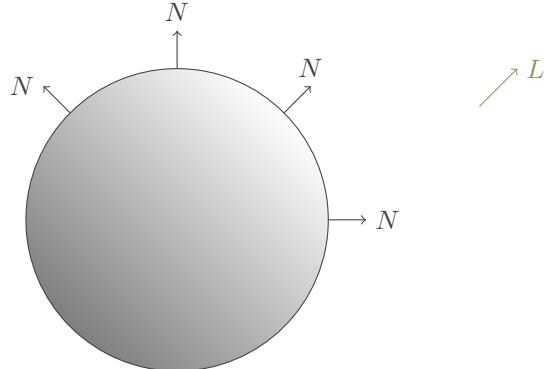


A priori, d'après Robert, pour avoir une jolie sphère en 3D, vous pourriez utiliser les concepts précédents pour calculer l'intersection à une sphère pour chaque rayon. Puis s'il y a intersection, calculer l'intensité d'une lumière directionnelle L (par exemple $L = \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}$) à l'aide de la normale de la surface N par un coefficient d'illumination simple :

$$\frac{\langle L \cdot N \rangle}{\|L\| \|N\|}$$

La normale pour une sphère se calcule depuis la position P sur la sphère et le centre C de la sphère par :

$$N = \frac{P - C}{\|P - C\|}$$



Ces explications ont achevé Charlie qui vous confie la mission de terminer son code.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 600;
const char * titre = "ESGI sphères";

const int sphères_count = 4;
double temps = 0;
const int pixel_size = 2;

double camera_position_x = 0.;
double camera_position_y = 0.;
double camera_position_z = -5.;

/* Implicit :
double camera_direction_x = 0. ;
double camera_direction_y = 0. ;
double camera_direction_z = 1. ;
*/
```

```
double sphere_intersection(double start_x, double start_y, double
→ start_z, double direction_x, double direction_y, double direction_z,
→ double center_x, double center_y, double center_z, double radius) {
    /* TODO : renvoyer le décalage t à appliquer (start + t * direction)
    → pour toucher la sphère de centre center et rayon radius */
    /* TODO : s'il n'y a pas intersection, renvoyer une grande valeur
    → négative. */
    return 0.;
}

double intensity(double x, double y, double z, double center_x, double
→ center_y, double center_z) {
    /* TODO : calculer normale (P - C) / norme(P - C) */
    /* TODO : définir une direction de lumière ambiante */
    /* TODO : calculer le produit scalaire de la normale et la direction de
    → la lumière */
    return 0.5;
}
```

```

int nearest(int first_sphere, double first_offset, int second_sphere,
→ double second_offset) {
    /* TODO : déterminer la sphère visible la plus proche */
    return (rand() % 2) ? first_sphere : second_sphere;
}

/* TODO : coder une couleur RGBA sur un entier */

unsigned int sphere_color(int id) {
    unsigned char r, g, b, a;
    a = 255;
    r = (id * 1337 + 127) % 256;
    g = (id * 133712 + 127) % 256;
    b = (id * 13371234 + 127) % 256;
    return 0 /* TODO : renvoyer la couleur RGBA */;
}

double sphere_x(int id) {
    return sin(id * 1274 + 0.2 * temps);
}

double sphere_y(int id) {
    return sin(id * 127478 + (0.3 + id * 0.1) * temps);
}

double sphere_z(int id) {
    return sin(id * 8899 + 0.5 * temps);
}

double sphere_r(int id) {
    return 0.375 + 0.25 * sin(id * 855) + 0.1 * sin(temps);
}

unsigned int pixel_color(int x, int y) {
    unsigned char r, g, b, a;
    double offset_x = x / (largeur / 2.) - 1.;
    double offset_y = y / (hauteur / 2.) - 1.;
    offset_x *= ((float)largeur / hauteur);

    int nearest_sphere;
    double nearest_t;
    int i;
    double t;
    for(i = 0; i < spheres_count; ++i) {
        t = sphere_intersection(camera_position_x + offset_x,
→ camera_position_y + offset_y, camera_position_z,

```

```
    0., 0., 1., sphere_x(i), sphere_y(i), sphere_z(i), sphere_r(i));  
    if(i == 0) {  
        nearest_sphere = i;  
        nearest_t = t;  
    } else {  
        nearest_sphere = nearest(nearest_sphere, nearest_t, i, t);  
        if(nearest_sphere == i) {  
            nearest_t = t;  
        }  
    }  
}  
  
if(nearest_t < 0.) {  
    return 0;  
}  
unsigned int color = sphere_color(nearest_sphere);  
/* TODO : récupérer la couleur RGB */  
r = 0;  
g = 0;  
b = 0;  
double value = intensity(camera_position_x + offset_x,  
    ↪ camera_position_y + offset_y, camera_position_z + nearest_t,  
    ↪ sphere_x(nearest_sphere), sphere_y(nearest_sphere),  
    ↪ sphere_z(nearest_sphere));  
a = 255;  
r = value * r;  
g = value * g;  
b = value * b;  
return 0 /* TODO : renvoyer la couleur RGBA */;  
}  
  
void affichage(SDL_Surface * ecran) {  
    /* Remplissage de l'écran par un gris foncé uniforme : */  
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));  
    /* Affichage du joueur : */  
    int x, y;  
    unsigned int color;  
    unsigned char r, g, b, a;  
    SDL_Rect pixel;  
    for(x = 0; x < largeur; x += pixel_size) {  
        for(y = 0; y < hauteur; y += pixel_size) {  
            color = pixel_color(x + pixel_size / 2., y + pixel_size / 2.);  
            /* TODO : récupérer la couleur RGB */  
            r = 0;  
            g = 0;
```

```

        b = 0;
        a = 0;
        if(a < 127) continue;
        pixel.x = x;
        pixel.y = y;
        pixel.w = pixel_size;
        pixel.h = pixel_size;
        SDL_FillRect(ecran, &pixel, SDL_MapRGB(ecran->format, r, g, b));
    }
}
}

int main() {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    int active = 1;
    SDL_Event event;
    int FPS_delay = 1000 / 60;
    int delay;

    while(active) {

        temps = SDL_GetTicks() / 1000.;
        delay = SDL_GetTicks();
        affichage(ecran);
        SDL_Flip(ecran);

        while(SDL_PollEvent(&event)) {

            switch(event.type) {
                /* Utilisateur clique sur la croix de la fenêtre : */
                case SDL_QUIT : {

```

```
    active = 0;
} break;

/* Utilisateur enfonce une touche du clavier : */
case SDL_KEYDOWN : {
    switch(event.key.keysym.sym) {
        /* Touche Echap : */
        case SDLK_ESCAPE : {
            active = 0;
        } break;
    }
} break;
}

delay = FPS_delay - (SDL_GetTicks() - delay);
if(delay > 0) {
    SDL_Delay(delay);
}
}

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

Exercice 37 (∞ Saut d'un personnage et suivi par caméra).

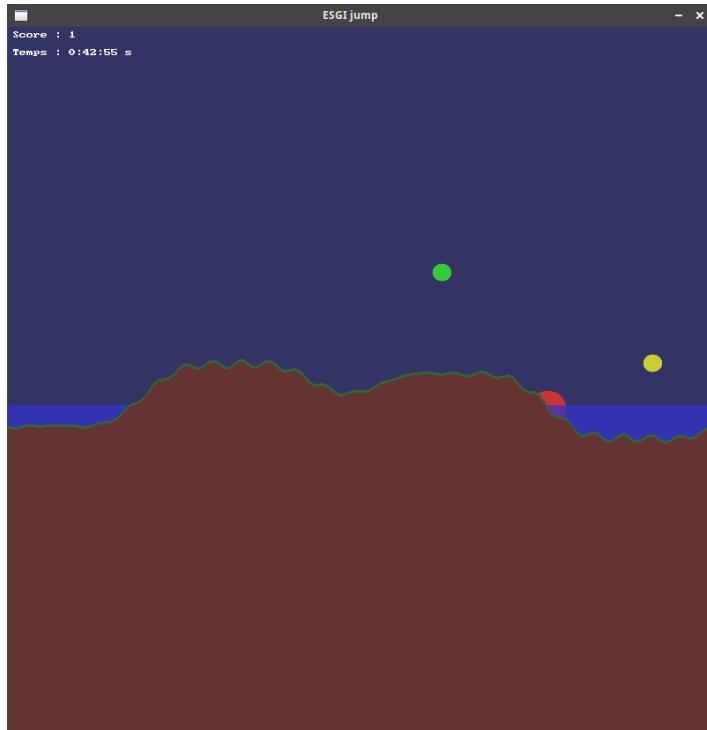
Oscar souhaite créer un jeu où le joueur pourrait se déplacer sur un terrain vu de côté. A priori, Charlie aurait trouvé la formule qui souhaite utiliser pour modéliser le terrain.

Dans son jeu, le joueur incarne une boule verte, doit collecter une boule jaune et éviter une boule rouge. Le joueur se dirige au clavier et peut sauter. S'il se trouve dans l'eau, la physique change. Oscar propose d'utiliser un modèle physique simple pour gérer le saut et la nage : basé sur position, vitesse et accélération.

$$\text{vitesse}_{i+1} \leftarrow \text{vitesse}_i + \text{accélération}_{i+1}$$

$$\text{position}_{i+1} \leftarrow \text{position}_i + \text{vitesse}_{i+1}$$

Il aimeraient que le jeu ressemble au moins à l'image suivante :



Le joueur serait déplacé sur les côtés en jouant sur la vitesse horizontale et une imagination de la gravité (saut) / poussée d'Archimède (eau) sur l'accélération verticale. Les impulsions de saut (par exemple de 60) et la gestion des forces constantes (gravité par exemple de 10 ou poussée d'Archimède par exemple de -1 en annulant la force de gravité) peuvent s'ajouter sur l'accélération.

Il vous a fait le code graphique et aimerait que vous complétiez les TODO avec des instructions (voir l'initialisation de SDL, section 15 dans le cours).

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI jump";

/* Coordonnées du joueur (position) : */
float px, py;
/* Vitesse du joueur (déplacement) : */
float vx, vy;
/* Accélération du joueur (saut, eau) : */
float ax, ay;

/* Coordonnées de l'objectif : */
float ox, oy;
/* Coordonnées de l'adversaire : */
float ex, ey;

/* Position de la caméra et échelle de la vue : */
float cx, cy, cz;

float distance(float first_x, float first_y, float second_x, float
    ↪ second_y) {
    float d = 0;
    /* TODO : calculer la distance entre deux position. */
    return d;
}

float hauteur_terrain(float x) {
    float h = 500 - x;
    /* TODO : proposer une modélisation du terrain. */
```

```
    return h;
}

void placer_objectif() {
    /* TODO : Placer l'objectif aléatoirement. */
    ox = 0;
    oy = 0;
}

void placer_adversaire() {
    /* TODO : Placer l'adversaire aléatoirement. */
    ex = 0;
    ey = 0;
}

int ecran_depuis_camera_x(float x) {
    /* TODO : Déterminer l'abscisse sur l'écran depuis une position sur le
       ↵ terrain. */
    return x + 100;
}

int ecran_depuis_camera_y(float y) {
    /* TODO : Déterminer l'ordonnée sur l'écran depuis une position sur le
       ↵ terrain. */
    return y + 100;
}

float camera_depuis_ecran_x(int x) {
    /* TODO : Déterminer l'abscisse sur le terrain depuis une position sur
       ↵ l'écran */
    return x - 100;
}

float camera_depuis_ecran_y(int y) {
    /* TODO : Déterminer l'ordonnée sur le terrain depuis une position sur
       ↵ l'écran */
    return y - 100;
}

float ecran_depuis_camera_z(int z) {
    /* TODO : Déterminer la mesure sur l'écran depuis sa mesure sur le
       ↵ terrain. */
    return z;
}
```

```

float camera_depuis_ecran_z(int z) {
    /* TODO : Déterminer la mesure sur le terrain depuis sa mesure sur
    → l'écran. */
    return z;
}

void affichage(SDL_Surface * ecran) {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));
    /* Affichage du joueur : */
    filledCircleRGBA(ecran, ecran_depuis_camera_x(px),
    → ecran_depuis_camera_y(py), ecran_depuis_camera_z(25), 51, 204, 51,
    → 255);

    filledCircleRGBA(ecran, ecran_depuis_camera_x(ox),
    → ecran_depuis_camera_y(oy), ecran_depuis_camera_z(25), 204, 204, 51,
    → 255);

    filledCircleRGBA(ecran, ecran_depuis_camera_x(ex),
    → ecran_depuis_camera_y(ey), ecran_depuis_camera_z(50), 204, 51, 51,
    → 255);

    int x, y, z;
    for(x = 0; x < largeur; ++x) {
        y = ecran_depuis_camera_y(hauteur_terrain(camera_depuis_ecran_x(x)));
        z = ecran_depuis_camera_y(0);
        boxRGBAlpha(ecran, x, y, x + 1, z, 51, 51, 204, 127);
        boxRGBAlpha(ecran, x, y, x + 1, hauteur, 102, 51, 51, 255);
        boxRGBAlpha(ecran, x, y, x + 1, y + ecran_depuis_camera_z(5), 51, 102,
        → 51, 255);
    }
}

int calculs(int score) {
    /* TODO : déplacer le joueur. */
    /* TODO : ajouter 1 au score si le joueur atteint l'objectif et le
    → replacer. */
    /* TODO : retirer 10 au score si le joueur est touché par l'adversaire
    → et le replacer. */
    /* TODO : déplacer l'adversaire. */
    return score;
}

void action_jump() {
    /* TODO : enclencher un saut. */
}

```

```
}

void action_droite() {
    /* TODO : déplacer le joueur vers la droite. */
}

void action_gauche() {
    /* TODO : déplacer le joueur vers la gauche. */
}

void action_sans_direction() {
    /* TODO : ne plus se diriger vers une direction. */
}

void deplacer_camera() {
    /* TODO : si le joueur sort ou est proche de sortir du champ de vision,
       → le suivre. */
}

int main() {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    /* Placement du joueur au centre de la fenêtre : */
    px = 0;
    py = 0;
    vx = 0;
    vy = 0;
    ax = 0;
    ay = 0;

    cx = 0;
    cy = 0;
```

```
cz = 500;

int active = 1;
SDL_Event event;
int moving_right = 0;
int moving_left = 0;
int jump = 0;

int score = 0;
unsigned int temps;
char display[100];

placer_objectif();

while(active) {

    temps = SDL_GetTicks();
    affichage(ecran);
    sprintf(display, "Score : %d", score);
    stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
    sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) / 60,
    → (temps / 1000) % 60, (temps % 1000) / 10);
    stringRGBA(ecran, 5, 25, display, 255, 255, 255, 255);
    SDL_Flip(ecran);

    while(SDL_PollEvent(&event)) {

        switch(event.type) {
            /* Utilisateur clique sur la croix de la fenêtre : */
            case SDL_QUIT : {
                active = 0;
            } break;

            /* Utilisateur enfonce une touche du clavier : */
            case SDL_KEYDOWN : {
                switch(event.key.keysym.sym) {
                    /* Touche Echap : */
                    case SDLK_ESCAPE : {
                        active = 0;
                    } break;

                    case SDLK_z :
                    case SDLK_UP : {
                        if(! jump) {
                            action_jump();
                        }
                    } break;
                }
            } break;
        }
    }
}
```

```
        }
        jump = 1;
    } break;

case SDLK_d :
case SDLK_RIGHT : {
    moving_right = 1;
} break;

case SDLK_q :
case SDLK_LEFT : {
    moving_left = 1;
} break;
}

} break;

case SDL_KEYUP : {
    switch(event.key.keysym.sym) {
        case SDLK_z :
        case SDLK_UP : {
            jump = 0;
        } break;

        case SDLK_d :
        case SDLK_RIGHT : {
            moving_right = 0;
        } break;

        case SDLK_q :
        case SDLK_LEFT : {
            moving_left = 0;
        } break;
    }
} break;

case SDL_MOUSEBUTTONDOWN : {
    switch(event.button.button) {
        case SDL_BUTTON_WHEELUP : {
            cz *= 0.8;
            if(cz < 1) {
                cz = 1;
            }
        } break;
    }
} break;

case SDL_BUTTON_WHEELDOWN : {
```

```
        cz /= 0.8;
        if(cz > 10000) {
            cz = 10000;
        }
    } break;
}
} break;
}

if(moving_right && moving_left) {
    action_sans_direction();
} else if(moving_right) {
    action_droite();
} else if(moving_left) {
    action_gauche();
} else {
    action_sans_direction();
}
score = calculs(score);

deplacer_camera();

if(score >= 10) {
    affichage(ecran);
    sprintf(display, "Score : %d", score);
    stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
    sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) / 60,
    ↳ (temps / 1000) % 60, (temps % 1000) / 10);
    stringRGBA(ecran, 5, 25, display, 255, 255, 255, 255);
    stringRGBA(ecran, largeur / 2 - 10, hauteur / 2, "BRAVO !", 255,
    ↳ 255, 255, 255);
    SDL_Flip(ecran);
    SDL_Delay(3000);
    active = 0;
}

SDL_Delay(1000 / 60);
}

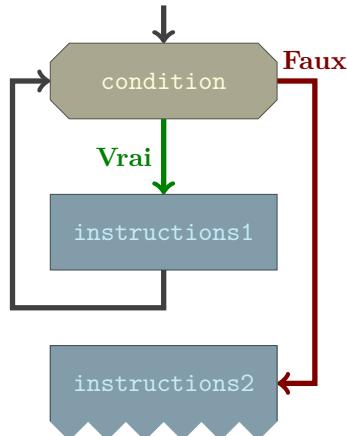
SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```


5 Boucles

Dans votre code, vous pouvez être amené à répéter un procédé jusqu'à arriver au résultat attendu. Pour ceci nous utiliserons des boucles. Les boucles permettent de lire des instructions et de relancer la lecture de celles-ci si besoin.

5.1 While : répétition sous condition

Une première boucle est la boucle `while` celle-ci fonctionne dans le même esprit que le `if` : si une condition est valide, on lit les instructions. La différence est que lorsque les instructions ont été lues, le `if` continue sur le code qui suit alors que la boucle `while` retourne vérifier si la condition est toujours vraie. Dans le cas où la condition est toujours vraie, elle exécute à nouveau les instructions qui la suivent. Elle se donne par la syntaxe suivante :



```
while(condition) {  
    /* instructions1 tant que condition est vraie */  
}  
/* instructions2 */
```

Par exemple, si on souhaite demander un nombre positif à un utilisateur, il est possible de lui redemander si sa saisie ne satisfait pas les attendus de notre programme :

```
int nombre = 0;  
printf("Entrez un nombre positif s'il-vous-plaît : ");  
scanf("%d", &nombre);  
while(nombre < 0) {  
    printf("Je répète, entrez un nombre positif s'il-vous-plaît :  
        ");
```

```

    scanf("%d", &nombre);
}
printf("Oh, un nombre positif : %d\n", nombre);

```

En sortie :

```

Entrez un nombre positif s'il-vous-plaît : -1
Je répète, entrez un nombre positif s'il-vous-plaît : -2
Je répète, entrez un nombre positif s'il-vous-plaît : 42
Oh, un nombre positif : 42

```

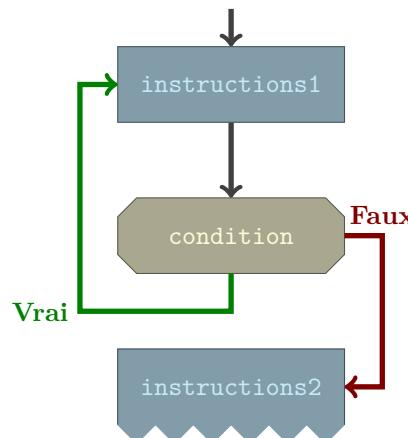
5.2 Do while : répétition si besoin

Pour le type d'utilisation vue précédemment, un autre type de boucle peut-être judicieux : la syntaxe `do while`, celle-ci permet de placer la condition à la fin : ceci veut dire que le code sera lu peu importe la condition puis répété tant la condition reste vraie. Sa syntaxe est la suivante :

```

do {
    /* instructions1 répétées tant que condition est vraie */
} while(condition);
/* instructions2 */

```



Dans l'exemple vu précédemment, ceci donnerait :

```
int nombre = 0;
do {
    printf("Entrez un nombre positif s'il-vous-plaît : ");
    scanf("%d", &nombre);
} while(nombre < 0);
printf("Oh, un nombre positif : %d\n", nombre);
```

En sortie :

```
Entrez un nombre positif s'il-vous-plaît : -1
Entrez un nombre positif s'il-vous-plaît : 42
Oh, un nombre positif : 42
```

5.3 For : un pas après l'autre

Souvent, nous utiliserons les boucles pour itérer sur des procédés et encore plus souvent pour compter de 0 à une valeur donnée. Ceci peut se faire avec une boucle `while` :

```
int compteur = 0;
while(compteur < 5) {
    printf("Le compteur vaut %d\n", compteur);
    ++compteur;
}
printf("Et voilà !\n");
```

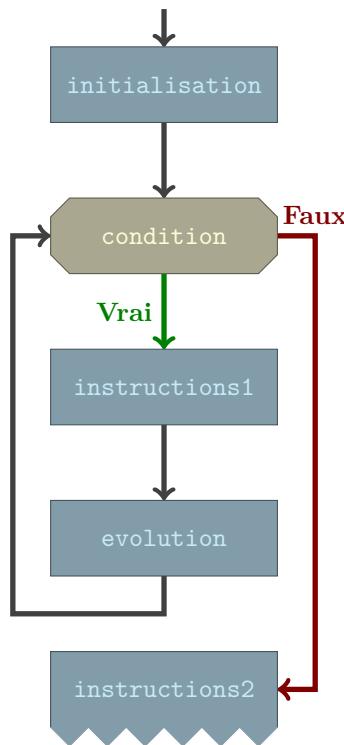
En sortie :

```
Le compteur vaut 0
Le compteur vaut 1
Le compteur vaut 2
Le compteur vaut 3
Le compteur vaut 4
Et voilà !
```

La réalité est que souvent la valeur donnée au compteur ou l'incrémentation du compteur seront oubliés ce qui conduira à des résultats étranges ou des boucles infinies. Une syntaxe qui convient à ce type de procédé est la boucle `for` : elle

embarque une initialisation, une condition comme la boucle `while` et le procédé d'itération. Sa syntaxe est donnée par :

```
for(initialisation; condition; evolution) {
    /* instructions1 tant que condition est vraie */
}
/* instructions2 */
```



Ceci fait que l'on peut utiliser la boucle `for` pour produire le même résultat que précédemment comme il suit :

```
int compteur;
for(compteur = 0; compteur < 5; ++compteur) {
    printf("Le compteur vaut %d\n", compteur);
}
printf("Et voilà !\n");
```

En sortie :

```
Le compteur vaut 0
Le compteur vaut 1
Le compteur vaut 2
Le compteur vaut 3
Le compteur vaut 4
Et voilà !
```

Une convention souvent utilisée sera de commencer à compter à partir de 0. Nous pouvons imbriquer des boucles par exemple pour dessiner un triangle :

```
int ligne, colonne;
for(ligne = 1; ligne <= 5; ++ligne) {
    for(colonne = 1; colonne <= ligne; ++colonne) {
        printf("*");
    }
    printf("\n");
}
```

En sortie :

```
*
**
***
****
*****
```

5.4 Contrôle de la boucle : break or continue

Il est possible de forcer l'interruption d'un boucle à l'aide du mot clé **break**. Par exemple si on demande un nombre positif, mais que l'utilisateur refuse de coopérer :

```
int nombre = 0;
int repetitions = 0;
do {
    printf("Entrez un nombre positif s'il-vous-plaît : ");
    scanf("%d", &nombre);
```

```

    ++repetitions;
    if(repetitions > 3) {
        printf("Ça suffit !\n");
        break;
    }
} while(nombre < 0);

if(nombre < 0) {
    printf("Ragequit.\n");
    exit(EXIT_FAILURE);
}
printf("Oh, un nombre positif : %d\n", nombre);
exit(EXIT_SUCCESS);

```

En sortie :

```

Entrez un nombre positif s'il-vous-plaît : -4
Entrez un nombre positif s'il-vous-plaît : -42
Entrez un nombre positif s'il-vous-plaît : -421
Entrez un nombre positif s'il-vous-plaît : -4210
Ça suffit !
Ragequit.

```

Il est aussi possible de forcer la boucle à se relancer depuis la vérification de la condition à l'aide du mot clé `continue`. Ceci est utile par exemple pour ne pas faire une partie des instructions de la boucle si une condition est vérifiée :

```

int nombre = 0;
int repetitions = 0;
do {
    printf("Entrez un nombre positif : ");
    scanf("%d", &nombre);
    ++repetitions;
    if(repetitions < 2) {
        continue;
    }
    printf("Entrez un nombre négatif : ");
    scanf("%d", &nombre);
    nombre *= -1;
} while(nombre < 0);

```

```
printf("Oh, un nombre positif : %d\n", nombre);
```

En sortie :

```
Entrez un nombre positif : -1
Entrez un nombre positif : -4
Entrez un nombre négatif : -42
Oh, un nombre positif : 42
```

5.5 Résumé

Boucle while : répète les instructions tant qu'une condition est valide :

```
while([CONDITION]) {  
    [INSTRUCTIONS]  
}
```

Boucle do while : exécute les instructions puis répète ces instructions tant qu'une condition est valide :

```
do {  
    [INSTRUCTIONS]  
} while([CONDITION]);
```

Boucle for : permet une initialisation, répète les instructions tant qu'une condition est valide et effectue une mise à jour à chaque fin d'exécution des instructions :

```
for([INITIALISATION]; [CONDITION]; [EVOLUTION]) {  
    [INSTRUCTIONS]  
}
```

Interruption d'une boucle :

```
[BOUCLE] {  
    break; /* sort de la boucle */  
}
```

Relance d'une boucle :

```
[BOUCLE] {  
    continue; /* retourne à la vérification de la condition*/  
}
```

5.6 Entraînement

Exercice 38 (★★ Liste des diviseurs).

Écrire un programme qui affiche la liste des diviseurs d'un nombre comme dans la sortie suivante :

```
Entrez un entier : 42
Liste des diviseurs de 42 :
1, 2, 3, 6, 7, 14, 21, 42
```

Exercice 39 (★★★ Force brute).

Alice trouve que Oscar fait trop le malin avec son programme de messages codés. Pour rappel, Oscar envoie des messages qu'il faut multiplier par $k = 2\ 015\ 201\ 261$ modulo $p = 4\ 285\ 404\ 239$. Robert vous indique que la clé secrète de Oscar multipliée avec k modulo p doit faire 1 et que sa méthode brute force lui trouve cette clé en environ 30 secondes. Proposez un programme qui trouve la clé secrète d'Oscar.

Exercice 40 (★★★ Affichage en binaire).

Bob aimerait demander un nombre entier à l'utilisateur et afficher sa représentation en binaire. Robert lui dit que soustraire les puissances de 2 depuis la plus grande possible permet de le faire. Voici un extrait du résultat du programme de Robert :

```
Entrez un nombre : 42
42 = (101010)_2
```

Exercice 41 (★★★ PGCD).

Charlie aimerait implémenter l'algorithme d'Euclide pour calculer le Plus Grand Diviseur Commun et en afficher les différentes étapes comme dans la sortie ci-dessous. Écrire le programme qui réalise cette sortie.

```
Entrez deux entiers : 1337 42
1337 = 42 * 31 + 35
42 = 35 * 1 + 7
35 = 7 * 5 + 0
pgcd(1337, 42) = 7
```

Exercice 42 (★★ Exponentiation d'un entier).

Alice souhaite calculer des puissances d'entiers, mais la fonction `pow` de la bibliothèque `cmath` utilise des nombres flottants. Elle vous missionne donc de coder un calcul de puissance pour des entiers. Ceci pourrait donner les sorties suivantes :

```
Entrez un entier : 10
Entrez un exposant : 10
10 puissance 10 = 10000000000
```

```
Entrez un entier : 2
Entrez un exposant : 60
2 puissance 60 = 1152921504606846976
```

```
Entrez un entier : 5
Entrez un exposant : 100
Cette exponentiation n'est pas assez triviale pour ce programme
→ (le résultat doit être sous 2 puissance 64)
```

```
Entrez un entier : 7
Entrez un exposant : -1
7 puissance -1 = 0
```

Exercice 43 (★★★ Jeu du plus ou moins).

Oscar vous propose de l'affronter à un jeu : il va choisir un nombre aléatoirement entre 0 et 1000 et vous devrez le deviner. Les seuls indications qu'il pourra vous donner est de savoir si le nombre que vous donnez est plus petit ou plus grand que le sien. Préparez un programme contre lequel jouer. Ceci peut par exemple donner la sortie suivante :

```
Nous avons choisi un nombre entre 0 et 1000
A quel nombre pensez-vous ? 500
Trop petit.
A quel nombre pensez-vous ? 750
Trop grand.
A quel nombre pensez-vous ? 625
Trop grand.
A quel nombre pensez-vous ? 562
Trop petit.
A quel nombre pensez-vous ? 593
Trop petit.
A quel nombre pensez-vous ? 609
Bien joué, le nombre était en effet 609
```

Robert vous propose le code suivant pour simuler de l'aléatoire :

05_robert.c : Code de Robert

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    srand(time(NULL));
    const int max = 1001;
    int nombre = rand() % max;
    /* À vous de jouer */
    exit(EXIT_SUCCESS);
}
```

Exercice 44 (★★★ Implémentation racine carré).

Bob pensait que calculer une racine carré $\sqrt{}$ était quelque chose d'assez trivial. Cependant Charlie lui explique que la machine ne sait pas calculer naturellement une racine carré et que nous pouvons implémenter la méthode de Newton pour calculer une racine carré. C'est une méthode qui permet de résoudre l'équation suivante pour f une fonction à valeurs réelles et x une inconnue :

$$f(x) = 0$$

Or, il se trouve que la racine carré d'un réel α peut s'exprimer comme solution d'une telle équation :

$$\begin{aligned}x &= \sqrt{\alpha} \\x^2 &= \sqrt{\alpha^2} \\x^2 &= |\alpha| \\x^2 - |\alpha| &= 0\end{aligned}$$

La méthode de Newton, nous donne ensuite que pour une estimation de départ x_0 , nous pouvons itérer vers une solution par les itérations suivantes :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

À noter f' la fonction dérivée de la fonction f . Elles sont données ici par les fonctions suivantes :

$$\begin{aligned}f : x &\mapsto x^2 - |\alpha| \\f' : x &\mapsto 2x\end{aligned}$$

Votre mission est d'implémenter la racine carré par la méthode donnée et comparer votre résultat et performances à la fonction proposée dans la bibliothèque `cmath`. Ceci pourrait par exemple donner les tests suivants :

```
Entrez un réel : 2
La racine carré de 2 est 1.4142135623730949 / 1.41421 (en 0.000013
    ↵   s)
La racine carré de 2 est 1.4142135623730951 / 1.41421 (en 0.000005
    ↵   s) (d'après math)
```

```
Entrez un réel : 1e99
La racine carré de 1e+99 est
→ 31622776601683792672080889218142074438676549468160.0000000000000
→ / 3.16228e+49 (en 0.000012 s)
La racine carré de 1e+99 est
→ 31622776601683792672080889218142074438676549468160.0000000000000
→ / 3.16228e+49 (en 0.000005 s) (d'après math)
```

```
Entrez un réel : 42
La racine carré de 42 est 6.4807406984078604 / 6.48074 (en
→ 0.000013 s)
La racine carré de 42 est 6.4807406984078604 / 6.48074 (en
→ 0.000005 s) (d'après math)
```

```
Entrez un réel : 1e-99
La racine carré de 1e-99 est 0.0000000000000000 / 3.16228e-50 (en
→ 0.000012 s)
La racine carré de 1e-99 est 0.0000000000000000 / 3.16228e-50 (en
→ 0.000005 s) (d'après math)
```

Exercice 45 (∞ Dessin de fractales).

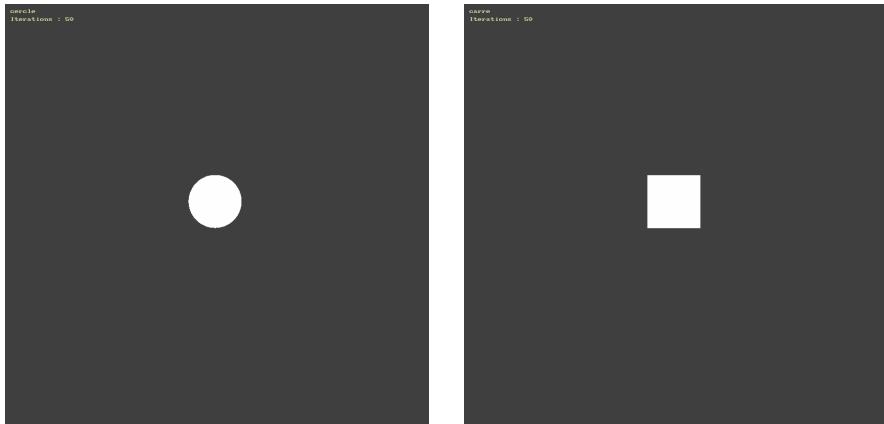
Oscar trouve que pour un débutant vous évoluez bien en programmation. Il a entendu parler de fractales connues et souhaite se lancer sur un projet avec vous pour les visualiser. Il souhaite définir quelques fonctions et pouvoir les visualiser. Dans un premier temps, vous pourriez essayer de dessiner des formes simples comme un cercle ou un rectangle :

- Un cercle de rayon R et de centre C est l'ensemble des points M tels que :

$$(C_x - M_x)^2 + (C_y - M_y)^2 \leq R^2$$

- Un carré de côté R et de centre C est l'ensemble des points M tels que :

$$\max(|C_x - M_x|, |C_y - M_y|) \leq \left(\frac{R}{2}\right)^2$$



Il souhaite ensuite dessiner des fractales connues dans l'informatique. Le souci est que ces fractales font intervenir des nombres complexes. Il a cependant trouvé les propriétés suivantes sur ces nombres :

- Les nombres complexes introduisent une dimension imaginaire par le nombre imaginaire i ayant la propriété d'être la solution de $\sqrt{-1}$ et donc étant donné par la propriété :

$$i^2 = -1$$

- Un nombre complexe $z \in \mathbb{C}$ peut s'écrire depuis deux réels $(x, y) \in \mathbb{R}^2$ tel que :

$$z = x + iy$$

- Nous pouvons séparer partie réelle x et imaginaire y d'un nombre complexe z par les fonctions partie réelle $Re(z)$ et partie imaginaire $Im(z)$:

$$z = Re(z) + i Im(z)$$

- L'addition de nombres complexes $z = x + iy$ et $w = a + ib$ est donnée par :

$$z + w = (x + a) + i(y + b)$$

- La multiplication de nombres complexes $z = x + iy$ et $w = a + ib$ est donnée par :

$$z * w = (xa - yb) + i(xb + ya)$$

- L'inverse de $z = x + iy$ peut s'exprimer par son conjugué $\bar{z} = x - iy$ et son module $|z| = \sqrt{x^2 + y^2}$ (par la propriété $z \times \bar{z} = |z|^2$, d'où) :

$$\frac{1}{z} = \frac{\bar{z}}{|z|^2}$$

- Nous pouvons associer à un nombre complexe $z = x + iy$ le point d'affixe (x, y) dans un plan.
(Oscar a été gentil et vous a déjà implémenté quelques opérations sur les nombres complexes).

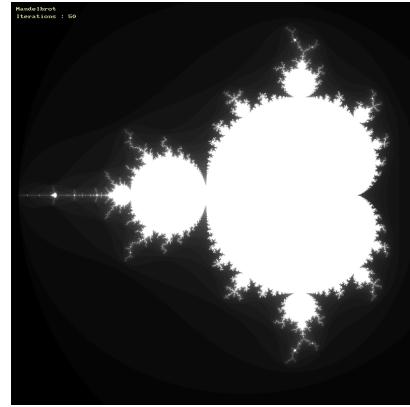
Des exemples connus que souhaite implémenter Oscar sont par exemple la fractale de Mandelbrot et l'ensemble de Julia (dont vous pourrez contrôler la position de la constante par les touches JLIK). Pour dessiner de telles fractales, vous devrez déterminer si une suite complexe $(z_n)_{n \in \mathbb{N}}$ ne diverge pas pour un nombre d'itérations données. La suite associée est donnée par une valeur initiale z_0 , une constante c et la relation suivante :

$$z_{n+1} = z_n^2 + c$$

Nous supposerons que cette suite ne diverge pas si pour toutes les itérations n données $|z_n| < 2$. Pour un pixel (x, y) , les paramètres sont les suivants :

- Fractale de Mandelbrot :

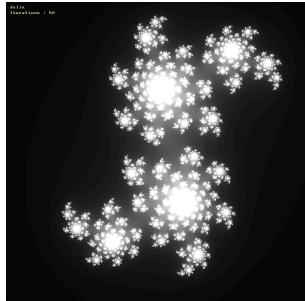
- $z_0 = 0$.
- $c = x + \mathbf{i}y$.



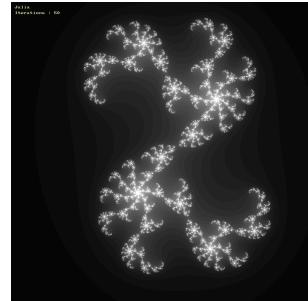
- Ensemble de Julia :

- $z_0 = x + \mathbf{i}y$.
- c une position variable choisie.

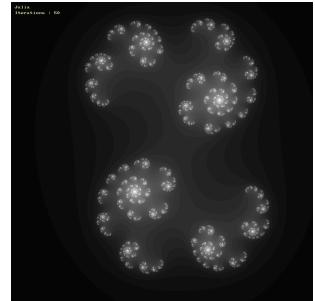
$$c = 0.35 + \mathbf{i}0.4$$



$$c = 0.4 - \mathbf{i}0.125$$



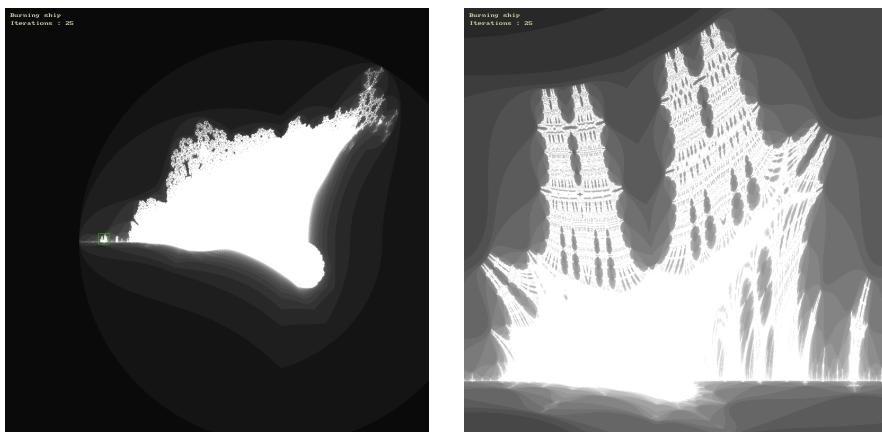
$$c = 0.4 - \mathbf{i}0.1$$



Une variation de cette suite permet d'obtenir la fractale du "Burning ship". Il suffit de reprendre les paramètres de la fractale de Mandelbrot et changer légèrement la formule d'itération en :

$$z_{n+1} = (|Re(z)| + \mathbf{i}|Im(z)|)^2 + c$$

Ce qui permet d'obtenir la fractale suivante (le Burning ship se trouve en bas à gauche) :



Oscar profite du fait que vous ayez entendu parler de la méthode de Newton pour vous préciser qu'elle peut être utilisée pour réaliser des fractales. En effet, selon la position initiale de la méthode, la solution peut différer. Il a donc choisi résoudre la racine n -ième. Celle-ci a n solutions qui vérifient l'équation suivante :

$$z^n = 1$$

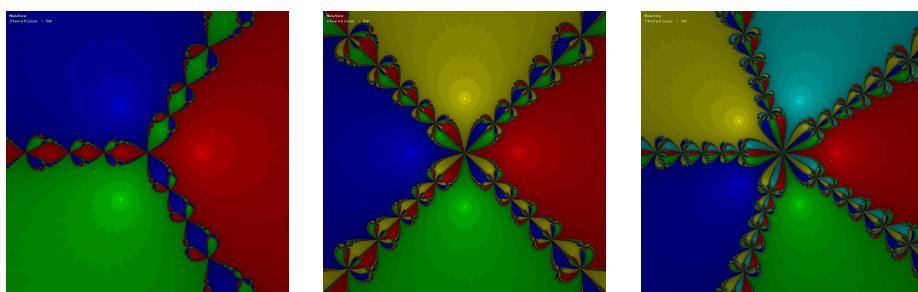
La formule d'itération est donc donnée par :

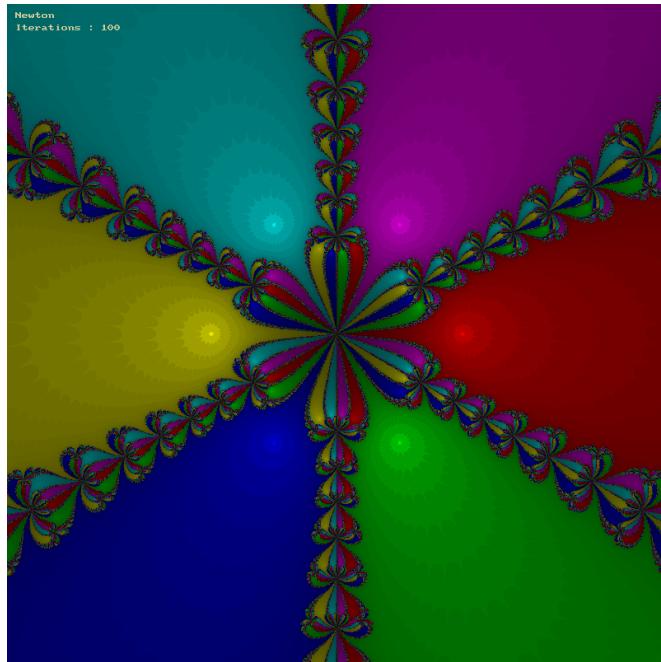
$$z_{k+1} = z_k - \frac{z_k^n - 1}{nz_k^{n-1}}$$

Les solutions sont les suivantes :

$$\left\{ \cos\left(\frac{2k\pi}{n}\right) + i \sin\left(\frac{2k\pi}{n}\right) \mid k \in [n] \right\}$$

En associant une couleur à la solution trouvée, ceci peut donner les fractales suivantes :





Voici un code de départ qu'il vous propose de compléter :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI draw";
const int functions_count = 8;
int resolution = 1;

double jx = 0, jy = 0;
int max_iter = 25;
int newton_n = 3;

/* Position de la caméra et échelle de la vue : */
float cx, cy, cz;
```

```
int current_function = 0;

int (* fonction(int id))(double x, double y);

const char * fonction_name(int id);

/* Multiplication de nombres complexes : */
/* utilisation : */
/* double ax, ay; */
/* double bx, by; */
/* double rx, ry; */
/* complex_mul(ax, ay, bx, by, &rx, &ry); */
void complex_mul(double a, double b, double x, double y, double * rx,
→ double * ry) {
    *rx = a * x - b * y;
    *ry = a * y + b * x;
}

/* Division de nombres complexes : */
/* utilisation : */
/* double ax, ay; */
/* double bx, by; */
/* double rx, ry; */
/* complex_div(ax, ay, bx, by, &rx, &ry); */
void complex_div(double a, double b, double x, double y, double * rx,
→ double * ry) {
    double norm = x * x + y * y;
    complex_mul(a, b, x, -y, rx, ry);
    *rx /= norm;
    *ry /= norm;
}

/* Puissance de nombres complexes : */
/* utilisation : */
/* double ax, ay; */
/* int n; */
/* double rx, ry; */
/* complex_pow(ax, ay, n, &rx, &ry); */
void complex_pow(double a, double b, int n, double * rx, double * ry) {
    *rx = 1;
    *ry = 0;
    while(n > 0) {
        if(n % 2 == 1) {
            complex_mul(*rx, *ry, a, b, rx, ry);
        }
        complex_mul(rx, ry, rx, ry, rx, ry);
        n /= 2;
    }
}
```

```
    }
    complex_mul(a, b, a, b, &a, &b);
    n /= 2;
}
}

int ecran_depuis_camera_x(float x) {
/* TODO */
return x;
}

int ecran_depuis_camera_y(float y) {
/* TODO */
return y;
}

float ecran_depuis_camera_z(int z) {
/* TODO */
return z;
}

float camera_depuis_ecran_x(int x) {
/* TODO */
return x;
}

float camera_depuis_ecran_y(int y) {
/* TODO */
return y;
}

float camera_depuis_ecran_z(int z) {
/* TODO */
return z;
}

const unsigned int R_CHANNEL = 0x01000000;
const unsigned int G_CHANNEL = 0x00010000;
const unsigned int B_CHANNEL = 0x00000100;
const unsigned int A_CHANNEL = 0x00000001;

int draw_distance_circle(double x, double y) {
/* TODO : calculer distance à un cercle */
return 0xff;
}
```

```
int draw_distance_square(double x, double y) {
    /* TODO : calculer distance à un carré */
    return 0xff;
}

int draw_circle(double x, double y) {
    /* TODO : colorer si intérieur d'un cercle */
    return 0xff;
}

int draw_square(double x, double y) {
    /* TODO : colorer si intérieur d'un carré */
    return 0xff;
}

int draw_mandelbrot(double x, double y) {
    /* TODO : dessiner la fractale de Mandelbrot. */
    /* Elle est donnée par la relation suivante : */
    /* z_(n + 1) = z_n * z_n + c */
    /* avec z_0 = 0, c = (x, y). */
    /* l'afficher si elle ne diverge pas */
    /* (vérifier sqrt(x * x + y * y) < 2) */
    return 0xff;
}

int draw_julia(double x, double y) {
    /* TODO : dessiner la fractale de Julia */
    /* Elle est donnée par la relation suivante : */
    /* z_(n + 1) = z_n * z_n + c */
    /* avec z_0 = (x, y), c = (jx, jy). */
    /* l'afficher si elle ne diverge pas */
    /* (vérifier sqrt(x * x + y * y) < 2) */
    return 0xff;
}

int draw_burning_ship(double x, double y) {
    /* TODO : dessiner la fractale du Burning ship */
    /* Elle est donnée par la relation suivante : */
    /* z_(n + 1) = (abs(Re(z_n)) + i abs(Im(z_n))) ** 2 + c */
    /* avec z_0 = (0, 0), c = (x, y). */
    /* l'afficher si elle ne diverge pas */
    /* (vérifier sqrt(x * x + y * y) < 2) */
    return 0xff;
}
```

```

int draw_newton(double x, double y) {
    int i;
    /* TODO : dessiner la fractale de la racine n-ème de l'unité */
    /* selon convergence de la méthode de Newton. */
    /* Elle est donnée par la relation suivante : */
    /* z_(i + 1) = z_i - (z_i ** n - 1) / (n z_i ** (n - 1)) */
    /* avec z_0 = (x, y). */
    /* la méthode converge vers une des solutions suivantes : */
    /* f(cos(2 k pi / n), sin(2 k pi / n)) pour k entier entre 0 et n - 1}
    ↵ */
    return 0xff;
}

void affichage(SDL_Surface * ecran) {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));
    unsigned int color;
    unsigned char r, g, b, a;
    SDL_Rect pixel;
    double x, y;
    int i, j;
    for(i = 0; i < largeur; i += resolution) {
        for(j = 0; j < hauteur; j += resolution) {
            x = ecran_depuis_camera_x(i + 0.5 * resolution);
            y = ecran_depuis_camera_y(j + 0.5 * resolution);
            color = fonction(current_function)(camera_depuis_ecran_x(i),
            ↵ camera_depuis_ecran_y(j));
            r = (color / R_CHANNEL) % 256;
            g = (color / G_CHANNEL) % 256;
            b = (color / B_CHANNEL) % 256;
            a = (color / A_CHANNEL) % 256;
            if(a < 127) continue;
            pixel.x = i;
            pixel.y = j;
            pixel.w = resolution;
            pixel.h = resolution;
            SDL_FillRect(ecran, &pixel, SDL_MapRGB(ecran->format, r, g, b));
        }
    }

    stringRGBA(ecran, 10, 10, fonction_name(current_function), 204, 204,
    ↵ 153, 255);
    char buffer[1024];
    sprintf(buffer, "Iterations : %d", max_iter);
}

```

```
stringRGBAlpha(ecran, 10, 25, buffer, 204, 204, 153, 255);  
}  
  
const char * fonction_name(int id) {  
    switch(id) {  
        case 1 : return "carre";  
        case 2 : return "distance cercle";  
        case 3 : return "distance carre";  
        case 4 : return "Mandelbrot";  
        case 5 : return "Julia";  
        case 6 : return "Burning ship";  
        case 7 : return "Newton";  
        default : return "cercle";  
    }  
}  
  
int (* fonction(int id))(double x, double y) {  
    switch(id) {  
        case 1 : return draw_square;  
        case 2 : return draw_distance_circle;  
        case 3 : return draw_distance_square;  
        case 4 : return draw_mandelbrot;  
        case 5 : return draw_julia;  
        case 6 : return draw_burning_ship;  
        case 7 : return draw_newton;  
        default : return draw_circle;  
    }  
}  
  
int main() {  
    srand(time(NULL));  
    /* Cr ation d'une fen tre SDL : */  
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {  
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());  
        exit(EXIT_FAILURE);  
    }  
    ecran = NULL;  
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |  
        SDL_DOUBLEBUF)) == NULL) {  
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());  
        SDL_Quit();  
        exit(EXIT_FAILURE);  
    }  
    SDL_WM_SetCaption(titre, NULL);
```

```
/* Placement du joueur au centre de la fenêtre : */

cx = 0;
cy = 0;
cz = 10;

int active = 1;
SDL_Event event;
int grab = 0;
int refresh = 1;

while(active) {

    if(refresh) {
        affichage(ecran);
        SDL_Flip(ecran);
        refresh = 0;
    }

    while(SDL_PollEvent(&event)) {

        switch(event.type) {
            /* Utilisateur clique sur la croix de la fenêtre : */
            case SDL_QUIT : {
                active = 0;
            } break;

            /* Utilisateur enfonce une touche du clavier : */
            case SDL_KEYDOWN : {
                switch(event.key.keysym.sym) {
                    /* Touche Echap : */
                    case SDLK_ESCAPE : {
                        active = 0;
                    } break;

                    case SDLK_j : {
                        jx -= 0.025;
                        refresh = 1;
                    } break;
                    case SDLK_l : {
                        jx += 0.025;
                        refresh = 1;
                    } break;
                    case SDLK_i : {
                        jy += 0.025;
                    } break;
                }
            }
        }
    }
}
```

```
    refresh = 1;
} break;
case SDLK_k : {
    jy -= 0.025;
    refresh = 1;
} break;

case SDLK_y : {
    max_iter += 5;
    refresh = 1;
} break;
case SDLK_h : {
    max_iter -= 5;
    refresh = 1;
} break;

case SDLK_t : {
    newton_n++;
    refresh = 1;
} break;
case SDLK_g : {
    newton_n--;
    refresh = 1;
} break;
}

} break;

case SDL_KEYUP : {
    switch(event.key.keysym.sym) {
        case SDLK_UP : {
            current_function = (current_function + 1) %
                functions_count;
            refresh = 1;
        } break;

        case SDLK_DOWN : {
            current_function = (current_function + functions_count - 1)
                % functions_count;
            refresh = 1;
        } break;
    }
} break;

case SDL_MOUSEBUTTONDOWN : {
    switch(event.button.button) {
```

```

        case SDL_BUTTON_WHEELUP : {
            cz *= 0.8;
            if(cz < 1e-9) {
                cz = 1e-9;
            }
            refresh = 1;
        } break;

        case SDL_BUTTON_WHEELDOWN : {
            cz /= 0.8;
            if(cz > 1e9) {
                cz = 1e9;
            }
            refresh = 1;
        } break;

        case SDL_BUTTON_LEFT : {
            grab = 1;
        } break;
    }

} break;

case SDL_MOUSEBUTTONDOWN : {
    switch(event.button.button) {
        case SDL_BUTTON_LEFT : {
            grab = 0;
        } break;
    }
} break;

case SDL_MOUSEMOTION : {
    if(grab) {
        cx += camera_depuis_ecran_z(-event.motion.xrel);
        cy += camera_depuis_ecran_z(-event.motion.yrel);
        refresh = 1;
    }
} break;
}

SDL_Delay(1000 / 60);

}

SDL_FreeSurface(ecran);
SDL_Quit();

```

```
    exit(EXIT_SUCCESS);
}
```


Troisième partie

Notions de base

Table des matières

6 Fonctions	189
6.1 La fonction main	189
6.2 Définition de fonctions	191
6.3 Portée des variables	194
6.3.1 Variables locales à une fonction	194
6.3.2 Variables globales	198
6.3.3 Portée des variables	198
6.4 Déplacer sa définition de fonction	200
6.4.1 Déclaration d'une fonction	200
6.4.2 Appels mutuels entre fonctions interdépendantes	201
6.5 Résumé	202
6.6 Entraînement	203
7 Tableaux	237
7.1 Tableau à une dimension	237
7.2 Chaînes de caractères	241
7.3 Tableau à plusieurs dimensions	242
7.4 Résumé	246
7.5 Entraînement	247
8 Pointeurs	261
8.1 Un type d'adresse	261
8.2 Arithmétique des pointeurs	263
8.3 Allocation dynamique	266
8.4 Résumé	269
8.5 Entraînement	271

6 Fonctions

Dans ce que nous avons présenté, nous laissons supposer que tout le code pouvait s'écrire dans les accolades suivant le `main()`. Cependant, plus le code grandit plus il devient difficile à lire, à maintenir et certains procédés se répètent. Un vrai casse-tête. En réalité, nous pouvons créer des procédés que nous pouvons utiliser sans avoir à les recoder à chaque utilisation. C'est le cas de `printf` et `scanf` que nous avions emprunté à la bibliothèque `stdio.h`. Ces procédures se nomment **fonctions**.

6.1 La fonction main

Pour définir une fonction, il faut lui donner un nom, un type de retour et des arguments par la syntaxe suivante :

```
typeDeRetour nomDeLaFonction (type1 argument1, type2 argument2,  
→ ... , typeN argumentN) {  
    /* instructions */  
}
```

Tout comme nous l'avions fait sans le savoir en définissant la fonction de `main`, de type de retour `int` et sans arguments :

```
int main() {  
    /* Nos instructions. */  
}
```

En réalité, la fonction `main` peut renvoyer une valeur à la fin à l'aide du mot clé `return` qui en précise la valeur au lieu d'utiliser `exit`. Par convention 0 est utilisé pour dire que tout s'est bien passé et une autre valeur est considérée comme un code d'erreur :

```
int main() {  
    /* Nos instructions. */  
    return 0;  
}
```

Cette fonction `main` peut en réalité aussi être définie avec des arguments :

```
int main(int argc, char * argv[]) {
    /* Nos instructions. */
}
```

Les arguments que nous pouvons donner à la fonction `main` permettent de récupérer données ajoutées lors du lancement du programme en ligne de commande ou si des fichiers sont glissés sur le programme pour l'ouvrir. `main_arguments.c` propose de visualiser les arguments reçus par la fonction.

main_arguments.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int nombre_arguments, char * valeurs_arguments[])
5 {
6     int i;
7     printf("%d arguments :\n", nombre_arguments);
8     for(i = 0; i < nombre_arguments; ++i) {
9         printf(" - \"%s\"\n", valeurs_arguments[i]);
10    }
11    exit(EXIT_SUCCESS);
12 }
```

En sortie :

```
# gcc -ansi -Wall -O2 -o main_arguments main_arguments.c
# ./main_arguments argument 1 "argument 2"
4 arguments :
- "./main_arguments"
- "argument"
- "1"
- "argument 2"
```

Note : Pour le moment, ne pas se focaliser sur le type `valeurs_arguments` ceci sera étudié dans les deux chapitres qui suivent.

6.2 Définition de fonctions

Nous pouvons aussi définir nos propres fonctions. Par exemple, une fonction `saluer` qui imprimera "Bonjour" dans la sortie de la console lorsqu'elle est appelée :

```
void saluer() {
    printf("Bonjour !\n");
}

int main() {
    saluer();
    saluer();
    exit(EXIT_SUCCESS);
}
```

En sortie :

```
Bonjour !
Bonjour !
```

Ici, son type de retour est `void` ce qui signifie que la fonction ne renvoie aucune valeur. Cependant une fonction peut renvoyer une valeur lorsqu'elle est appelée. C'est par exemple le cas de `printf` qui renvoie le nombre de caractères imprimés :

```
int retour_printf;
retour_printf = printf("123456789\n");
printf("printf avait imprimé %d caractères\n", retour_printf);
```

En sortie :

```
123456789
printf avait imprimé 10 caractères
```

De même `scanf` renvoie le nombre de formats pour lesquels la lecture a été réussie. À noter que `scanf` peut imposer une format que l'utilisateur doit respecter :

```
int first, second;
int count;
printf(">>> ");
```

```
while(scanf("%d + %d", &first, &second) == 2) {
    printf("%d\n", first + second);
    printf(">>> ");
}
printf("Terminé\n");
```

En sortie :

```
>>> 5 + 7
12
>>> 5 * 7
Terminé
```

Avec `scanf` nous avions pu rencontrer des problèmes lorsqu'un utilisateur peu coopératif ne respecte pas le format attendu. Avec les outils que nous avons étudié ensemble, nous pouvons maintenant gérer le code problématique suivant :

```
int age = -1;
while(age < 0) {
    printf("Entrez votre age : ");
    scanf("%d", &age);
}
printf("Vous avez donc %d ans\n", age);
```

En sortie :

```
Entrez votre age : NON
Entrez votre age : Entrez votre age : Entrez votre age : Entrez
←   votre age : Entrez votre age : Entrez votre age : Entrez votre
←   age : Entrez votre age : Entrez votre age : ...
```

Une possibilité est de vider les caractères présents jusqu'au retour à la ligne envoyé par l'utilisateur pour envoyer son entrée :

```
int age = -1;
while(age < 0) {
    printf("Entrez votre age : ");
    if(scanf("%d", &age) != 1) {
        while(getchar() != '\n');
    }
}
```

```
}
```

```
printf("Vous avez donc %d ans\n", age);
```

En sortie :

```
Entrez votre age : NON
Entrez votre age : Euh, tu ne veux plus crasher ?
Entrez votre age : Vraiment 1111111111 ?????
Entrez votre age : Bon, OK
Entrez votre age : 42
Vous avez donc 42 ans
```

Nous pouvons nous-mêmes construire des fonctions avec un type de retour comme par exemple pour effectuer des opérations :

```
int addition(int first, int second) {
    return first + second;
}

int main() {
    printf("addition de 1 et 2 = %d\n", addition(1, 2));
    exit(EXIT_SUCCESS);
}
```

En sortie :

```
addition de 1 et 2 = 3
```

Exercice 46 (★ Définir une fonction).

Définir une fonction carré telle que permette au code suivant de fonctionner :

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : fonction carre qui prend un int en entrée et renvoie un
   int en sortie */

int main() {
    int a = 41;
```

```

printf("(%d + 1) * (%d + 1) = %d\n", a, a,
(a + 1) * (a + 1));
printf("carre(%d + 1) = %d\n", a, carre(a + 1));
if((a + 1) * (a + 1) == carre(a + 1))
    printf("Bravo !\n");
return 0;
}

```

6.3 Portée des variables

Une variable est déclarée et utilisable dans la section correspondante à cette déclaration. Jusqu'ici nous avons déclaré des variables dans la fonction main. Nous ne nous sommes donc pas soucié de la porté : espace sur lequel elle reste définie.

6.3.1 Variables locales à une fonction

Fonction main

Il est possible de déclarer une variable à tout moment dans la fonction main et de l'utiliser autant que l'on peut le souhaiter dans la fonction main à la suite de cette déclaration.

```

int main() {
    int variableDeMain = 42;
    printf("%d\n", variableDeMain);
    float autreVariableDeMain = 13.37;
    printf("%g\n", autreVariableDeMain);
    variableDeMain = 1;
    printf("%d\n", variableDeMain);
    exit(EXIT_SUCCESS);
}

```

Cependant, cette variable ne peut pas être utiliser en dehors de la fonction main :

```

void test() {
    variableDeMain = 10;
}

int main() {
    int variableDeMain = 42;
}

```

```
printf("%d\n", variableDeMain);
test();
printf("%d\n", variableDeMain);
exit(EXIT_SUCCESS);
}
```

Le code suivant provoquera en effet une erreur de compilation :

```
# gcc -o prog main.c
main.c: In function ‘test’:
main.c:5:2: error: ‘variableDeMain’ undeclared (first use in this
→   function)
variableDeMain = 10;
^~~~~~
main.c:5:2: note: each undeclared identifier is reported only once
→   for each function it appears in
```

Paramètres d'une fonction

Pour passer une valeur à une fonction, il est donc possible de lui envoyer par argument :

```
void test(int variableDeMain) {
    variableDeMain = 10;
    printf("%d\n", variableDeMain);
}

int main() {
    int variableDeMain = 42;
    printf("%d\n", variableDeMain);
    test(variableDeMain);
    printf("%d\n", variableDeMain);
    exit(EXIT_SUCCESS);
}
```

Ce qui donnerait la sortie suivante :

```
42
10
42
```

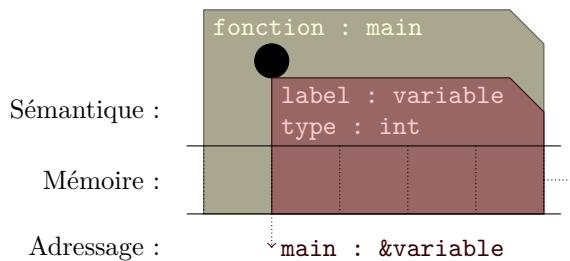
Notons ici que bien que les deux variables aient le même nom, ceci ne correspond pas au même emplacement mémoire :

```
void test(int variable) {
    variable = 10;
    printf("%p : %d\n", &variable, variable);
}

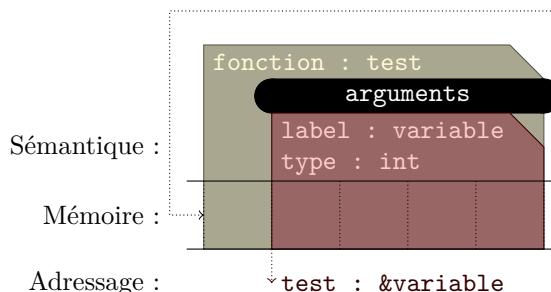
int main() {
    int variable = 42;
    printf("%p : %d\n", &variable, variable);
    test(variable);
    printf("%p : %d\n", &variable, variable);
    exit(EXIT_SUCCESS);
}
```

Ce qui donnerait la sortie suivante :

```
0x7ffc773564b4 : 42
0x7ffc7735649c : 10
0x7ffc773564b4 : 42
```



En effet, la fonction `test` déclare une variable `variableDeMain` en paramètre et lorsque `test` est appelée, c'est la valeur passée en argument qui est copiée comme lors d'une affectation.

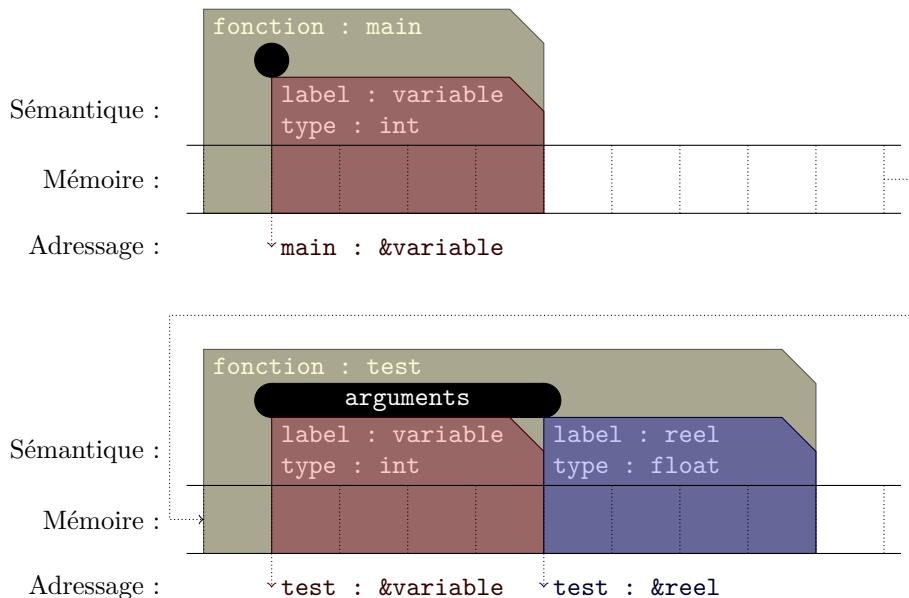


Variables locales

Tout comme la fonction `main`, tout fonction peut définir des variables qui lui sont locales autres que ses paramètres.

```
void test(int variable) {
    float reel = 13.37;
    variable = 10;
    printf("%p : %d\n", &variable, variable);
    printf("%p : %g\n", &reel, reel);
}

int main() {
    int variable = 42;
    printf("%p : %d\n", &variable, variable);
    test(variable);
    printf("%p : %d\n", &variable, variable);
    exit(EXIT_SUCCESS);
}
```



6.3.2 Variables globales

Cependant de rares éléments peuvent avoir l'intérêt d'être partagés dans tout le code. Pour ceci, il est possible de déclarer une variable globale : avant le champs de fonctions. Le code suivant définit une variable globale que chaque fonction peut utiliser :

```
int variable;

void test() {
    variable = 10;
    printf("%p : %d\n", &variable, variable);
}

int main() {
    variable = 42;
    printf("%p : %d\n", &variable, variable);
    test();
    printf("%p : %d\n", &variable, variable);
    exit(EXIT_SUCCESS);
}
```

Ce qui donnerait la sortie suivante :

```
0x55f64c9e2014 : 42
0x55f64c9e2014 : 10
0x55f64c9e2014 : 10
```

Ceci reste à utiliser de manière raisonnée. Par exemple, partager la fenêtre de son interface graphique dans tout le code sans avoir à le passer à chaque fonction en argument est pertinent. Cependant, partager avec tout le code une opérande d'un calcul particulier l'est moins et vous demandera de produire plus de code. Utilisez les variables globales que lorsque nécessaire.

6.3.3 Portée des variables

Nous avons vu que les variables définies en argument d'une fonction ou dans son bloc de définition lui sont propres et ne sont visibles que pour la fonction. Ceci vient du fait qu'à l'exécution, lors de l'appel à la fonction, on empile ces variables (on leur fait une place en RAM dans la pile) et lorsque le bloc associé à la fonction se termine, on dépile ces variables (on libère la place qu'elles occupaient dans la

RAM). La portée des variables locales aux fonctions ayant le même nom peut se représenter comme il suit :

```
void test() {
    int variable = 10;
    printf("%p : %d\n", &variable, variable);
}

int main() {
    int variable = 42;
    printf("%p : %d\n", &variable, variable);
    test();
    printf("%p : %d\n", &variable, variable);
    exit(EXIT_SUCCESS);
}
```

Dans le cas où une variable globale existe avec le même nom, c'est la variable locale qui prend la priorité :

```
int variable;

void test() {
    int variable = 10;
    printf("%p : %d\n", &variable, variable);
}

int main() {
    int variable = 42;
    printf("%p : %d\n", &variable, variable);
    test();
    printf("%p : %d\n", &variable, variable);
    exit(EXIT_SUCCESS);
}
```

Il est aussi possible de déclarer des variables directement dans des blocs. Dans ce cas, elles existent dans le code jusqu'à la fin du bloc. Ces variables prennent la priorité si elles sont contenues dans des blocs pour lesquels une variable du même nom est définie :

```

void test() {
    int variable = 10;
    printf("%p : %d\n", &variable, variable);
    if(variable == 10) {
        int variable = 42;
        printf("%p : %d\n", &variable, variable);
    }
    printf("%p : %d\n", &variable, variable);
    {
        int variable = 1337;
        printf("%p : %d\n", &variable, variable);
    }
    printf("%p : %d\n", &variable, variable);
}

```

6.4 Déplacer sa définition de fonction

6.4.1 Déclaration d'une fonction

Une fonction peut être déclarée à l'avance, avant même d'être définie. Ceci permet de placer la déclaration de la fonction avant son appel et sa définition après et où souhaité dans le code :

```

int addition(int, int);

int main() {
    printf("addition de 1 et 2 = %d\n", addition(1, 2));
    exit(EXIT_SUCCESS);
}

int addition(int first, int second) {
    return first + second;
}

```

En sortie :

```
addition de 1 et 2 = 3
```

À noter que le nommage des variables en argument n'est pas obligatoire dans la déclaration, mais la déclaration et la définition doivent avoir la même signature : le même nom et les mêmes types de retour et d'arguments.

6.4.2 Appels mutuels entre fonctions interdépendantes

Il se peut que dans le code, on souhaite faire que deux fonctions s'appellent mutuellement comme une fonction `ping` qui appelle `pong` et réciproquement. Ceci ne sera possible qu'en déclarant les fonctions avant de les définir :

```
void tic(int);
void tac(int);

void tic(int nombre) {
    if(nombre > 0) {
        printf("> tic !\n");
        tac(nombre - 1);
    }
}

void tac(int nombre) {
    if(nombre > 0) {
        printf("< tac !\n");
        tic(nombre - 1);
    }
}

int main() {
    tic(4);
    exit(EXIT_SUCCESS);
}
```

En sortie :

```
> tic !
< tac !
> tic !
< tac !
```

6.5 Résumé

Définition d'une fonction (définit la procédure à exécuter lors de son appel) :

```
[TYPE DE RETOUR] [NOM DE LA FONCTION] ([ARGUMENT 1], [ARGUMENT 2],
→ ... , [ARGUMENT N]) {
    [INSTRUCTIONS]
}
```

Déclaration d'une fonction (autorise son appel dans le code qui suit) :

```
[TYPE DE RETOUR] [NOM DE LA FONCTION] ([ARGUMENT 1], [ARGUMENT 2],
→ ... , [ARGUMENT N]);
```

Appel de la fonction :

```
[NOM DE LA FONCTION] ([VALEUR 1], [VALEUR 2], ..., [VALEUR N]);
```

Le mot-clé `return` termine les instructions dans le corps d'une fonction et renvoie la valeur associée (si le type de retour n'est pas `void`).

Exemple :

```
/* Déclaration : */
int addition(int, int);

/* Définition : */
int addition(int first, int second) {
    int résultat = first + second;
    return résultat;
}

int main() {
    /* Appel : */
    printf("addition de 1 et 1 : %d\n", addition(1, 1));
    exit(EXIT_SUCCESS);
}
```

6.6 Entraînement

Exercice 47 (** Fonction de menu).

Écrire une fonction `menu` qui continue tant que l'utilisateur n'a pas entré un nombre qui soit 1, 2 ou 3 et renvoie le choix de l'utilisateur de sorte de l'intégrer au code suivant :

01_menu.c

```
#include <stdio.h>
#include <stdlib.h>

int menu() {
    /* À vous de jouer */
}

int main() {
    printf("%d, bien reçu.\n", menu());
    exit(EXIT_SUCCESS);
}
```

Une utilisation peut être illustrée par la sortie suivante :

```
Menu :
 1 - un truc
 2 - un autre
 3 - quitter
---
Votre choix : 5
Menu :
 1 - un truc
 2 - un autre
 3 - quitter
---
Votre choix : 1
1, bien reçu.
```

Exercice 48 (★★ Calcul moyenne).

Compléter le code c-dessous en écrivant la fonction `moyenne` qui lit des flottants positifs depuis le clavier et en calcule la moyenne comme dans la sortie qui suit :

02_moyenne.c

```
#include <stdio.h>
#include <stdlib.h>

float moyenne();

int main() {
    printf("La moyenne de ");
    printf("= %g\n", moyenne());
    exit(EXIT_SUCCESS);
}
```

```
La moyenne de 12 16 15 -1
= 14.3333
```

```
La moyenne de 10 -42
= 10
```

```
La moyenne de 10 11 12 15 17 20 8 2 11 19 10 2 3 7 14 -1
= 10.7333
```

Exercice 49 (★★ Utilisation variables locales).

Bob essaie d'utiliser des variables locales dans son switch. Cependant, ceci ne semble pas fonctionner. Proposez une solution pour qu'il puisse conserver ses noms de variables mais que ça compile :

1. Une première version sans utiliser de fonctions.
2. Une seconde version en utilisant des fonctions.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int first, second;
    char res;
    printf(">>> ");
    scanf("%d %c %d", &first, &res, &second);
    switch(res) {
        case '+':
            int res = first + second;
            printf(" = %d\n", res);
            break;

        case '*':
            long res = (long)first * second;
            printf(" = %ld\n", res);
            break;

        case '/':
            double res = (double)first / second;
            printf(" = %lg\n", res);
            break;
    }
    exit(EXIT_SUCCESS);
}
```

Exercice 50 (★★★ Compléter fonctions).

Compléter le code suivant sans modifier la partie déjà codée :

```
#include <stdio.h>
#include <stdlib.h>

/* Debut de votre code : */
/* Fin de votre code. */

void compute(int first, char op, int second) {
    switch(op) {
        case '+':
            printf(" = %ld\n", addInt(first, second));
            break;
        case '-':
            printf(" = %ld\n", subInt(first, second));
            break;
        case '*':
            printf(" = %ld\n", mulInt(first, second));
            break;
        case '/':
            printf(" = %lg\n", divInt(first, second));
            break;
        case '%':
            printf(" = %d\n", modInt(first, second));
            break;
        case '^':
            printf(" = %ld\n", powInt(first, second));
            break;
    }
}

int main() {
    int first, second;
    char op;
    printf(">>> ");
    first = scanInt();
    op = scanOp();
    second = scanInt();
    compute(first, op, second);
    exit(EXIT_SUCCESS);
}
```

Exercice 51 (★★★ Calculatrice avec mémoire : séparation en fonctions).

Libérez la fonction `main` et remaniez le code suivant pour plus de lisibilité et avec une meilleure découpe en fonctions :

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int first, second;
    int variable = 0;
    char op;
    int choix;
    int resultat;
    do {
        printf("1 - Calculer\n2 - Modifier variable\n3 - Voir
        → variable\n0 - Quitter\n---\nVotre choix : ");
        scanf("%d", &choix);
        switch(choix) {
            case 1 :
                printf(">>> ");
                scanf("%d %c %d", &first, &op, &second);
                switch(op) {
                    case '+' : resultat = first + second; break;
                    case '-' : resultat = first - second; break;
                    case '*' : resultat = first * second; break;
                    case '/' : resultat = first / second; break;
                    case '%' : resultat = first % second; break;
                }
                printf(">>> %d %c %d = %d\n", first, op, second,
                → resultat);
                break;
            case 2 :
                printf(">>> variable = %d ", variable);
                scanf(" %c %d", &op, &second);
                switch(op) {
                    case '+' : variable += second; break;
                    case '-' : variable -= second; break;
                    case '*' : variable *= second; break;
                    case '/' : variable /= second; break;
                    case '%' : variable %= second; break;
                    case '=' : variable = second; break;
                }
        }
    } while(choix != 0);
```

```
    printf(">>> variable = %d\n", variable);
    break;
case 3 :
    printf(">>> variable = %d\n", variable);
    break;
default : break;
}
} while(choix != 0);
printf("Au revoir.\n");
exit(EXIT_SUCCESS);
}
```

Exercice 52 (★★★ Exponentiation et performances).

- Écrire une fonction `puissance` qui calcule a^n pour a et n deux entiers en respectant la déclaration suivante :

```
long puissance(long a, long n);
```

- Écrire une fonction `puissance_rapide` qui calcule a^n à l'aide de cet algorithme :

Algorithm 1: Exponentiation rapide

```
input : Entier valeur
input : Entier exposant
output: Entier resultat

1 resultat ← 1 ;
2 while exposant > 0 do
3   if exposant modulo 2 = 1 then
4     | resultat ← resultat × valeur ;
5   end
6   valeur ← valeur × valeur ;
7   exposant ← exposant/ 2;
8 end
```

- Écrire une fonction `puissance_modulo` et `puissance_rapide_modulo` qui calculent $a^n \bmod p$ pour a , n et p trois entiers en respectant les déclarations suivantes :

```
unsigned long puissance_modulo(unsigned long a, unsigned long
→ n, unsigned long p);
unsigned long puissance_rapide_modulo(unsigned long a,
→ unsigned long n, unsigned long p);
```

- Comparez les temps qu'il faut à ces deux fonctions pour calculer $42^{2\ 460\ 320\ 538} \bmod 4\ 285\ 404\ 239$.

Exercice 53 (★★★ Affrontement de personnages).

Proposez un code qui permet à deux personnages de s'affronter en tour par tour comme proposé par l'illustration de sortie suivante :

```
+-----+-----+
|     Joueur      |     Adversaire    |
+-----+-----+
| Vie :          100 | Vie :          100 |
| Attaque :       50 | Attaque :       50 |
| Defense :      20 | Defense :      20 |
+-----+-----+
1 - cogner.
2 - se soigner.
3 - augmenter attaque.
4 - augmenter defense.
Votre choix : 1
Le joueur cogne l' adversaire.
L'adversaire se soigne.
+-----+-----+
|     Joueur      |     Adversaire    |
+-----+-----+
| Vie :          100 | Vie :          100 |
| Attaque :       50 | Attaque :       50 |
| Defense :      20 | Defense :      20 |
+-----+-----+
1 - cogner.
2 - se soigner.
3 - augmenter attaque.
4 - augmenter defense.
Votre choix : 3
Le joueur augmente son attaque.
L'adversaire cogne le joueur.
+-----+-----+
|     Joueur      |     Adversaire    |
+-----+-----+
| Vie :          98  | Vie :          100 |
| Attaque :       60  | Attaque :       50 |
| Defense :      20  | Defense :      20 |
+-----+-----+
1 - cogner.
2 - se soigner.
3 - augmenter attaque.
4 - augmenter defense.
Votre choix :
```

Exercice 54 (★★★ Racine carré par dichotomie).

Le théorème des valeurs intermédiaires indique que si un fonction f est continue, strictement monotone sur un intervalle et que son signe change entre les deux bornes de l'intervalle, alors elle admet une solution c sur celui-ci pour $f(c) = 0$. Cette solution peut se déterminer par une recherche dichotomique de la solution : l'idée est de diviser l'intervalle de recherche par 2 en fonction du signe de la valeur médiane de la fonction sur l'intervalle. Implémenter cette recherche dichotomique pour résoudre la racine carré d'un réel α à l'aide de la fonction suivante :

$$\begin{aligned} f &: [0, \alpha] \rightarrow \mathbb{R} \\ x &\mapsto x^2 - |\alpha| \end{aligned}$$

Alice se rappelle avoir vu la recherche dichotomique en cours d'algorithmique et vous donne l'algorithme qui lui a été donné :

Algorithm 2: Recherche dichotomique à itérations fixées

```

input : Réel debut
input : Réel fin
input : Fonction fonction
input : Entier maxIterations
output: Réel solution

1 Pour maxIterations itérations faire
2   | solution  $\leftarrow \frac{\text{debut} + \text{fin}}{2}$  ;
3   | Si fonction (solution)  $\times$  fonction (debut)  $< 0$  Alors
4   |   | fin  $\leftarrow$  solution ;
5   | Fin
6   | Sinon si fonction (solution)  $\times$  fonction (fin)  $< 0$  Alors
7   |   | debut  $\leftarrow$  solution ;
8   | Fin
9   | Sinon
10  |   | Sortie de boucle ;
11  | Fin
12 Fin
13 Si  $|\text{debut} - \text{fin}| > \epsilon$  Alors
14   |   Arrêter l'algorithme avec erreur ;
15 Fin
16 /* solution est telle que fonction (solution) = 0 */
```

Bob, a lui déjà implémenté cette méthode et a comparé la méthode à la fonction native de la bibliothèque `cmath` :

```
Entrez un réel : 2
La racine carré de 2 est 1.4142135623730949 (en 0.000020 s)
La racine carré de 2 est 1.4142135623730951 (en 0.000005 s) (d'après
↪  math)
```

```
Entrez un réel : 42
La racine carré de 42 est 6.4807406984078604 (en 0.000009 s)
La racine carré de 42 est 6.4807406984078604 (en 0.000005 s) (d'après
↪  math)
```

```
Entrez un réel : 1e30
La racine carré de 1e+30 est 1000000000000000.0000000000000000 (en
↪  0.000012 s)
La racine carré de 1e+30 est 1000000000000000.0000000000000000 (en
↪  0.000005 s) (d'après math)
```

```
Entrez un réel : 1e99
La racine carré de 1e+99 est
↪  31622776601683792672080889218142074438676549468160.0000000000000000
↪  (en 0.000015 s)
La racine carré de 1e+99 est
↪  31622776601683792672080889218142074438676549468160.0000000000000000
↪  (en 0.000005 s) (d'après math)
```

```
Entrez un réel : 1e-9
La racine carré de 1e-09 est 0.0000316227766017 (en 0.000009 s)
La racine carré de 1e-09 est 0.0000316227766017 (en 0.000004 s) (d'après
↪  math)
```

Exercice 55 (★★★ Calcul coefficients binomiaux).

Bob a découvert les coefficients binomiaux. Un outil intéressant en combinatoire et donc en probabilités par exemple. A priori, on pourrait les calculer directement depuis leur formule récursive pour tous entiers naturels n et k :

$$\binom{n}{k} = \begin{cases} 0 & \text{Si } k < 0 \text{ ou } k > n \\ 1 & \text{Si } k = 0 \text{ ou } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{Sinon} \end{cases}.$$

Charlie l'informe qu'il peut aussi les calculer depuis une formule équivalente et bien plus efficace si bien codée. Ceci implique une écriture avec la suite $(n!)_{n \in \mathbb{N}}$ nommée factorielle définie telle que pour tout entier naturel n :

$$n! = \begin{cases} 1 & \text{Si } n \leq 1 \\ (n-1)! \times n & \text{Sinon} \end{cases}.$$

Notons que, pour tous entiers naturels n et k , nous avons

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Implémentez le calcul des coefficients binomiaux d'une manière récursive puis optimisée depuis l'écriture avec la fonction factorielle. Puis afficher le triangle de Pascal pour jusqu'à un nombre choisi par l'utilisateur :

```
Entrez l'entier n : 10
0 : 1
1 : 1 1
2 : 1 2 1
3 : 1 3 3 1
4 : 1 4 6 4 1
5 : 1 5 10 10 5 1
6 : 1 6 15 20 15 6 1
7 : 1 7 21 35 35 21 7 1
8 : 1 8 28 56 70 56 28 8 1
9 : 1 9 36 84 126 126 84 36 9 1
10 : 1 10 45 120 210 252 210 120 45 10 1
```

Exercice 56 (★★★ Développement de Taylor : calcul de sinus).

Bob trouve l'implémentation de la fonction sinus un peu lente sur son système. Il a découvert le développement en série de Taylor : ceci permet d'approcher une fonction en un point a par un polynôme à l'aide d'un calcul de ses dérivées successives $f^{(k)}$ et d'une compensation par une fonction R_n négligeable à proximité de a :

$$f(x) = \sum_{k=0}^{k=n} \frac{(x-a)^k}{k!} f^{(k)}(a) + R_n(x)$$

Bob est noyé dans ces mathématiques, Charlie décide donc de lui expliquer le cas pour la fonction sinus :

- La dérivée de sinus est cosinus.
- La dérivée de cosinus est le négatif de sinus.
- Les fonctions s'alternent donc par dérivées successives.

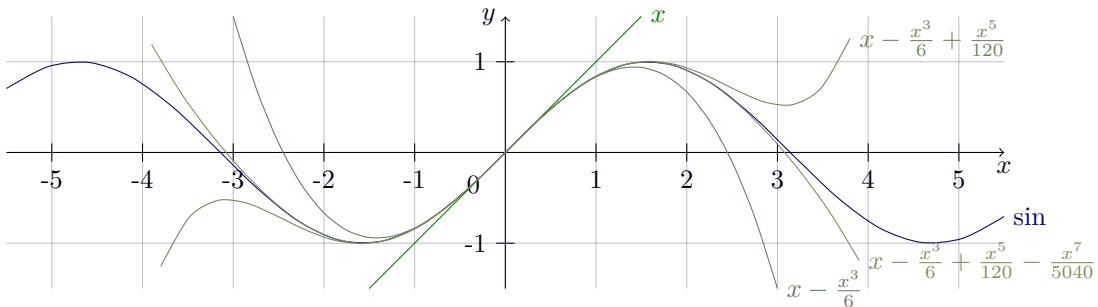
D'où un développement de Taylor suivant :

$$\begin{aligned} f(x) &= \sin(a) + \cos(a)(x-a) - \sin(a) \frac{(x-a)^2}{2} - \cos(a) \frac{(x-a)^3}{6} \\ &\quad + \sin(a) \frac{(x-a)^4}{24} + \cos(a) \frac{(x-a)^5}{120} + \dots \end{aligned}$$

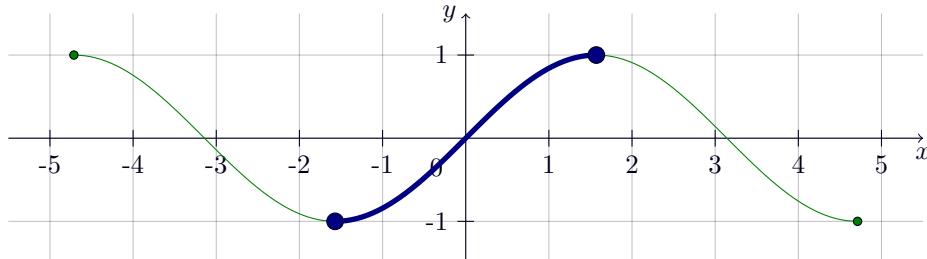
Charlie explique à Bob, qu'il pourrait tester des développements avec un choix de a astucieux :

- Pour $a = 0$, $\sin(a) = 0$ et $\cos(a) = 1$. D'où :

$$f(x) = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \dots$$



De plus, Charlie précise que bien que le polynôme n'épouse la forme de la fonction sinus qu'en voisinage de 0, la fonction sinus est périodique et symétrique : un morceau est donc suffisant pour approcher la fonction sinus en tout point :



Charlie indique à Bob qui a utilisé le polynôme précédemment donné et que ceci semble plus rapide que sa propre fonction native. Il arrive aussi à battre les performances de cette même fonction avec une plus grande précision s'il continue le développement jusqu'au degré 15 et en utilisant des `long double` :

```
Entrez un réel : 1
sin(1) = 0.8414682539682540 (en 0.000006 s) (rapide en 0)
sin(1) = 0.8414709848078965 (en 0.000057 s) (d'après math)
sinl(1) = 0.8414709848078937 (en 0.000005 s) (développé en 0)
sinl(1) = 0.8414709848078965 (en 0.000039 s) (d'après math)
```

```
Entrez un réel : 3.1415
sin(3.1415) = 0.0000926535898673 (en 0.000007 s) (rapide en 0)
sin(3.1415) = 0.0000926535896605 (en 0.000072 s) (d'après math)
sinl(3.1415) = 0.0000926535896603 (en 0.000005 s) (développé en 0)
sinl(3.1415) = 0.0000926535896605 (en 0.000040 s) (d'après math)
```

```
Entrez un réel : 3.141592653589793
sin(3.14159) = 0.000000000002069 (en 0.000006 s) (rapide en 0)
sin(3.14159) = 0.0000000000000001 (en 0.000086 s) (d'après math)
sinl(3.14159) = -0.0000000000000001 (en 0.000005 s) (développé en
→ 0)
sinl(3.14159) = 0.0000000000000001 (en 0.000031 s) (d'après math)
```

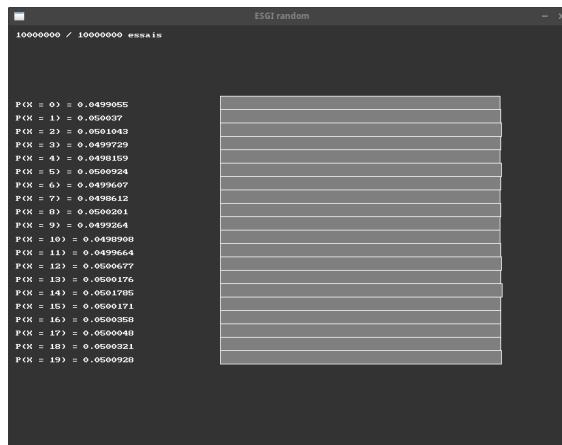
```
Entrez un réel : 42
sin(42) = -0.9165112513706832 (en 0.000006 s) (rapide en 0)
sin(42) = -0.9165215479156338 (en 0.000083 s) (d'après math)
sinl(42) = -0.9165215479156005 (en 0.000023 s) (développé en 0)
sinl(42) = -0.9165215479156338 (en 0.000031 s) (d'après math)
```

```
Entrez un réel : -2
sin(-2) = -0.9092884586394411 (en 0.000006 s) (rapide en 0)
sin(-2) = -0.9092974268256817 (en 0.000060 s) (d'après math)
sinl(-2) = -0.9092974268256550 (en 0.000005 s) (développé en 0)
sinl(-2) = -0.9092974268256817 (en 0.000039 s) (d'après math)
```

Après avoir vu que pour des résultats très analogues, sa fonction sinus pourrait aller de l'ordre de 10 fois plus vite que la fonction native, Bob est motivé pour l'implémenter dans son nouveau jeu vidéo !

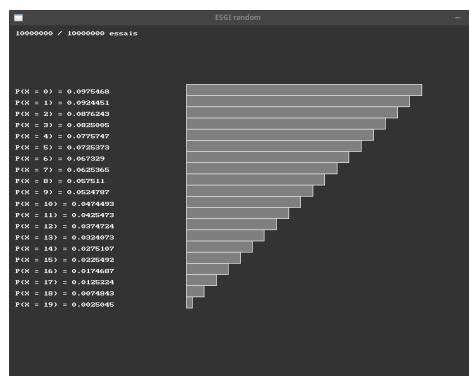
Exercice 57 (∞ Contrôle de l'aleatoire).

Charlie développe un nouveau RPG. Il aimeraient jouer avec l' aléatoire et avoir un peu de contrôle sur celui-ci. Pour ceci, il a créé un petit système pour vérifier la distribution de ses valeurs aléatoires. En effet, d'après la loi des grands nombres, pour un nombre suffisamment grand de répétition de l'expérience aléatoire, les fréquences observées convergent vers les probabilités de chaque issue. Il suppose que si la fonction est efficace, il devrait attendre quelques secondes pour vérifier la loi de probabilité associée à sa fonction d' aléatoire pour 10 000 000 d'essais. Par défaut, le générateur aléatoire standard propose une distribution uniforme des valeurs :

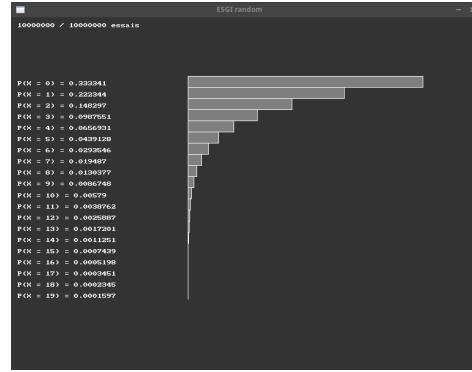


Il souhaite réussir à modéliser des générateurs aléatoires pouvant simuler les ré-partitions suivantes :

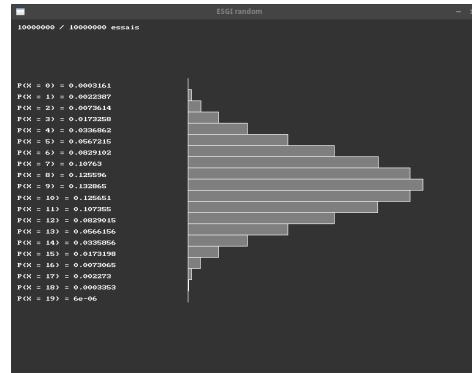
- Une répartition triangulaire uniforme :



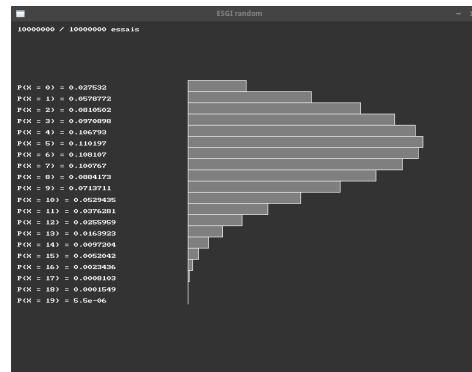
- Une répartition type loi exponentielle :



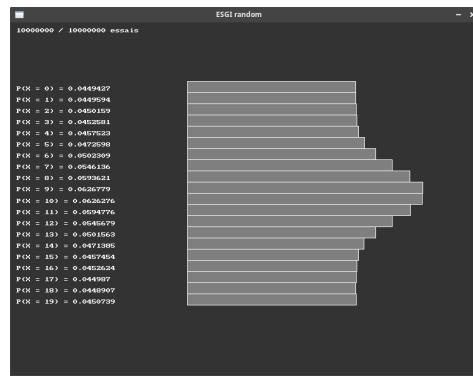
- Une répartition type loi normale :



- Une répartition type loi gamma :



- Une répartition hybride entre plusieurs lois :



Pour vous aider dans votre mission, voici de quoi visualiser la distribution de vos générateurs aléatoires :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

#define MAX_ENTRIES 20

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 600;
const char * titre = "ESGI random";
SDL_Surface * ecran = NULL;

const int trials = 10000000;
int essai = 0;
unsigned long counts[MAX_ENTRIES];

void draw_random() {

    char buffer[1024];
    int i;
    unsigned long max_count = 0;
    unsigned long sum_count = 0;
    for(i = 0; i < MAX_ENTRIES; ++i) {
        max_count = (max_count > counts[i]) ? max_count : counts[i];
        sum_count += counts[i];
    }
    if(max_count != 0)
        for(i = 0; i < MAX_ENTRIES; ++i)
            counts[i] /= sum_count;
    SDL_FillRect(ecran, NULL, 0);
    for(i = 0; i < MAX_ENTRIES; ++i)
        draw_bar(buffer, i, counts[i]);
    SDL_Flip(ecran);
}
```

```

        sum_count += counts[i];
    }
    for(i = 0; i < MAX_ENTRIES; ++i) {
        boxRGBA(ecran,
            300, 100 + (i * (hauteur - 200)) / (MAX_ENTRIES + 1),
            300 + ((largeur - 400) * (counts[i])) / max_count, 100 + ((i + 1) *
                → (hauteur - 200)) / (MAX_ENTRIES + 1),
            127, 127, 127, 255);
        rectangleRGBA(ecran,
            300, 100 + (i * (hauteur - 200)) / (MAX_ENTRIES + 1),
            300 + ((largeur - 400) * (counts[i])) / max_count, 100 + ((i + 1) *
                → (hauteur - 200)) / (MAX_ENTRIES + 1),
            255, 255, 255, 255);
        sprintf(buffer, "P(X = %d) = %g", i, (double)counts[i] / sum_count);
        stringRGBA(ecran, 10, 100 + ((i + 0.5) * (hauteur - 200)) /
            → (MAX_ENTRIES + 1), buffer, 255, 255, 255, 255);
    }
    sprintf(buffer, "%d / %d essais", essai, trials);
    stringRGBA(ecran, 10, 10, buffer, 255, 255, 255, 255);
}

int uniform_random(int entries) {
    return rand() % entries;
}

int baised_fair_take_random(int entries) {
    int i;
    for(i = 0; i < entries; ++i) {
        if(rand() % entries == 0) return i;
    }
    return entries - 1;
}

/* TODO : vos fonctions */

void test_random(int entries, int (*random)(int)) {
    char buffer[1024];
    int i;
    for(i = 0; i < MAX_ENTRIES; ++i) {
        counts[i] = 0;
    }
    unsigned long t = 1;
    int updates = 0;
    for(essai = 0; essai < trials; ++essai) {
        ++(counts[random(entries)]);
    }
}

```

```

if(essai > t || essai % 10000 == 0) {
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
    draw_random();
    SDL_Flip(ecran);
    if(essai > t) {
        t *= 2;
        SDL_Delay(1 + 1000 / (++updates));
    }
}
}

int read_command() {
    printf("Aléatoires :\n");
    printf(" 1. uniform random\n");
    printf(" 2. baised fair take random\n");
    /* TODO : lister vos fonctions */
    printf(" 0. quitter\n");
    printf(">>> ");
    int choix;
    scanf("%d", &choix);
    switch(choix) {
        case 0 : return 0;
        case 1 : test_random(MAX_ENTRIES, uniform_random); break;
        case 2 : test_random(MAX_ENTRIES, baised_fair_take_random); break;
        /* TODO : lister l'appel de vos fonctions */
        default : break;
    }
    return 1;
}

int main() {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);
}

```

```
SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
SDL_Flip(ecran);

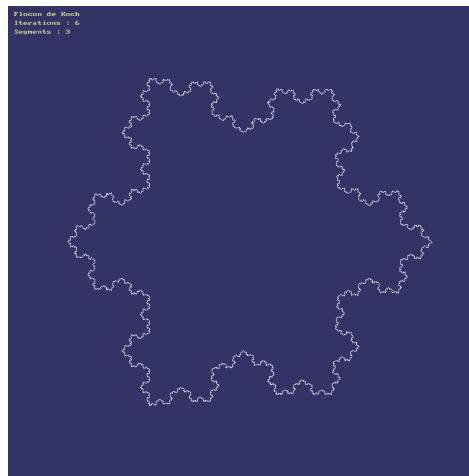
while(read_command()) {

    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
    draw_random();
    SDL_Flip(ecran);
    SDL_Delay(1000 / 60);
}

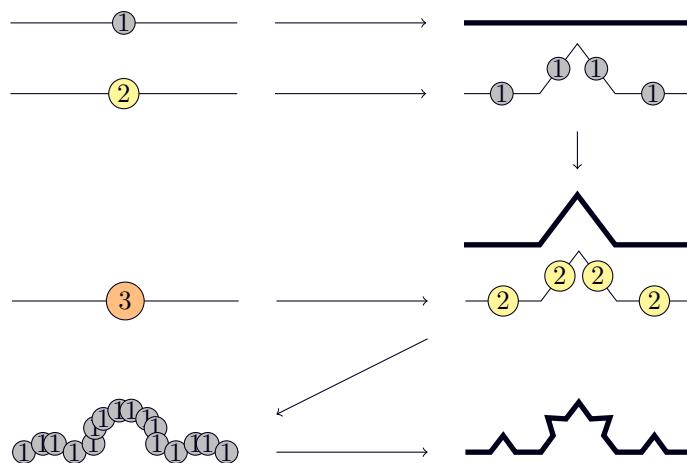
SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

Exercice 58 (∞ Courbes récursives).

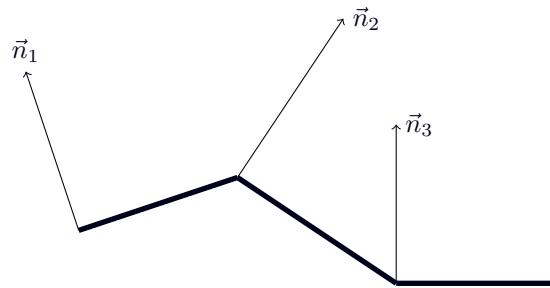
Oscar a découvert le concept de fonctions et par la même occasion qu'il pouvait les utiliser de manière récursive. C'est-à-dire que la fonction s'appelle elle-même. Ceci est pratique par exemple pour tracer des courbes selon un modèle comme le flocon de Koch :



Le concept est que nous travaillons sur une ligne et un degré répétition. Dans l'idée le dessin d'une ligne est une ligne de degré 1. Une ligne de degré 2 se construit depuis des lignes de degré 1. Une ligne de degré 3 se construit depuis des lignes de degré 2 et ainsi de suite :

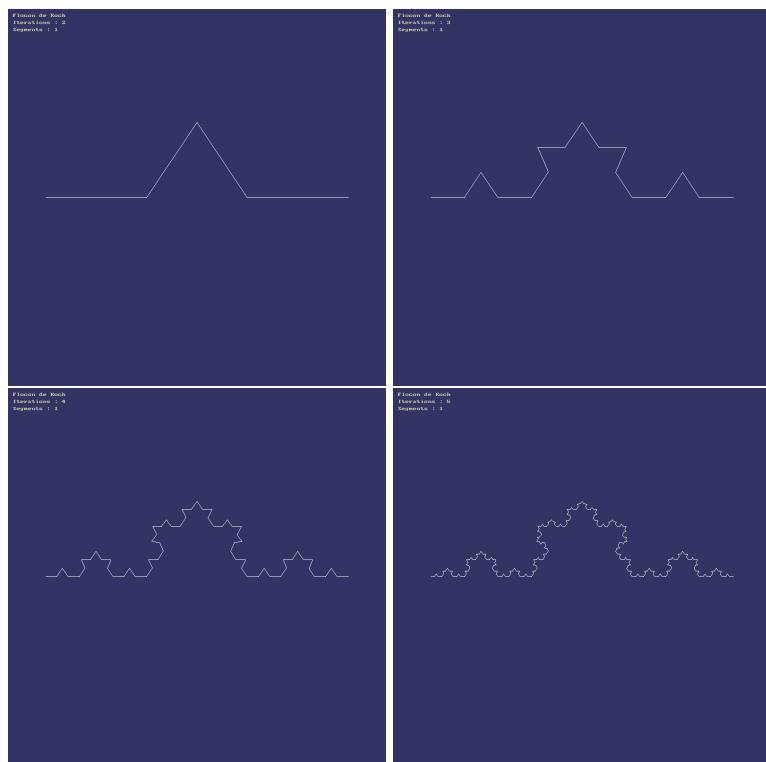


À noter que pour une ligne $(S_x, S_y) - (E_x, E_y)$, il est possible de définir une normale $\vec{n} = (E_y - S_y, S_x - E_x)$: vecteur perpendiculaire. En utilisant cette ligne et cette normale, nous pouvons définir un repère orthogonal et donc y représenter des coordonnées :

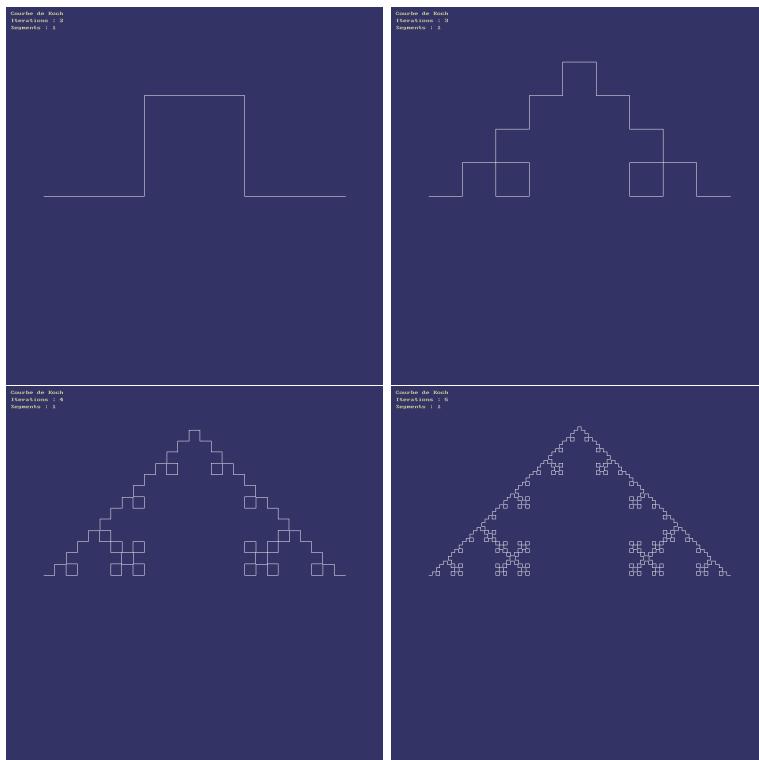


Depuis cette idée, vous pourriez implémenter les courbes récursives suivantes :

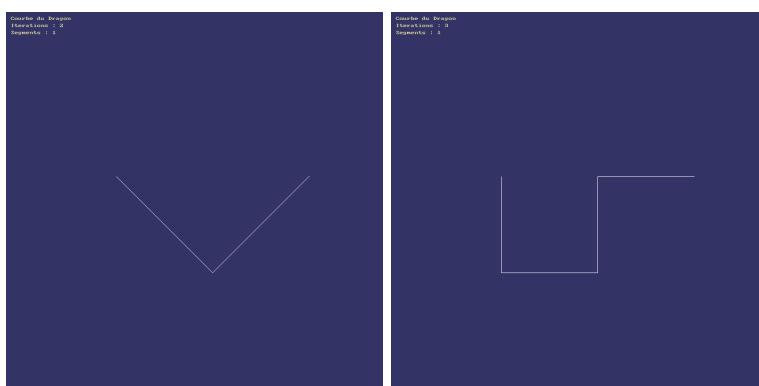
- Flocon de Koch :

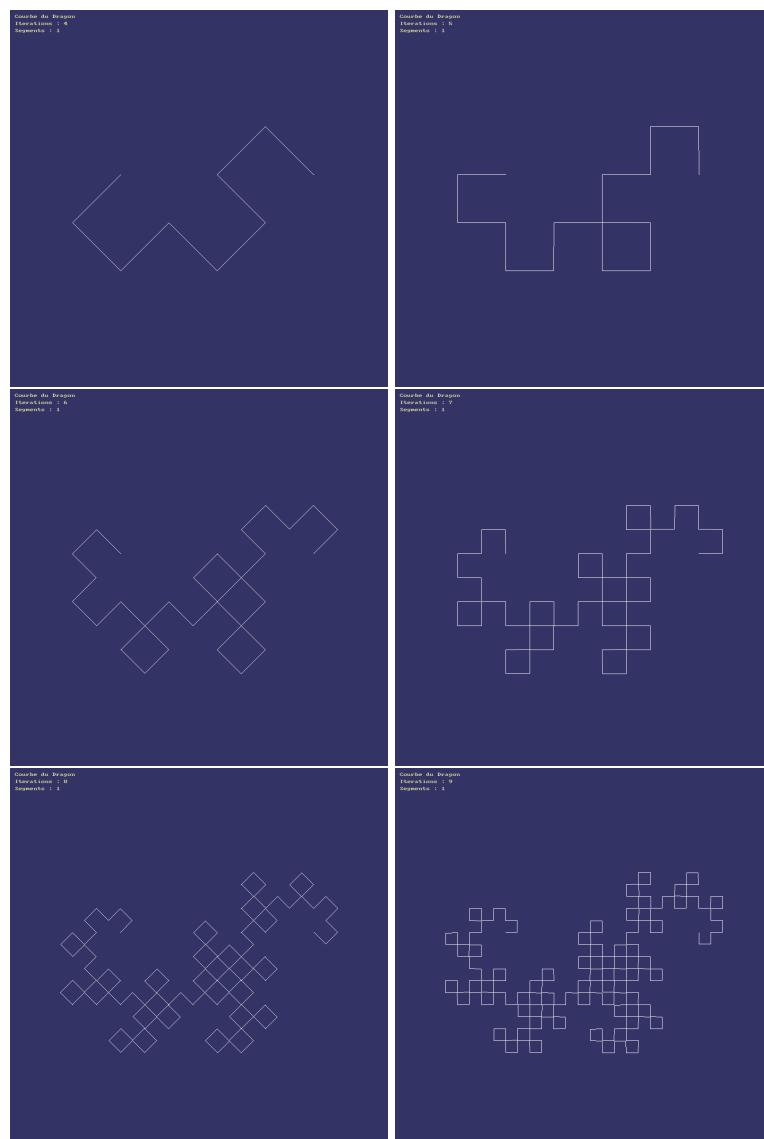


- Courbe de Koch :



- Courbe du dragon (changement de directions sur les lignes) :

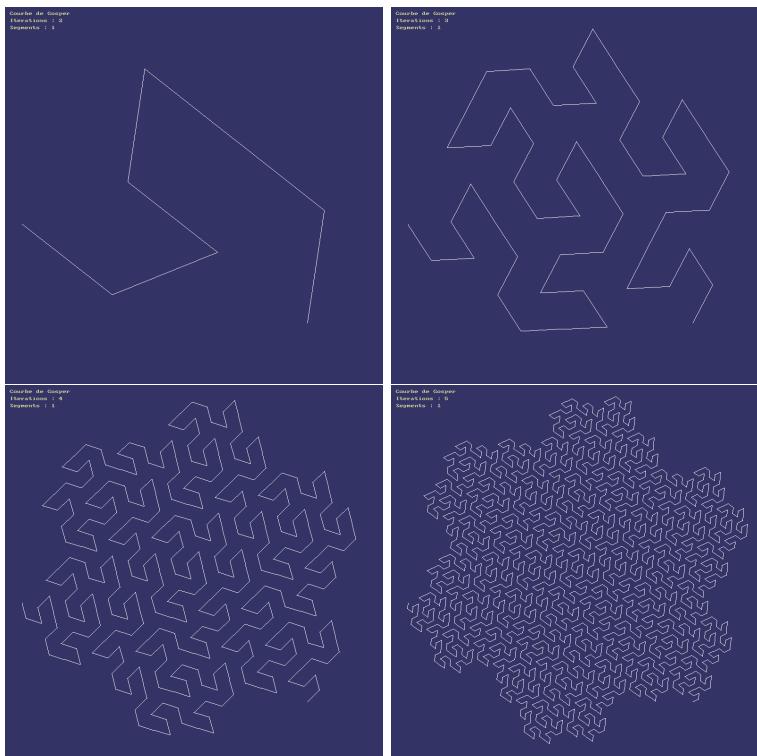




Pour aller plus loin, vous avez la possibilité de mettre en place un L-système : c'est l'idée de représenter un schéma récursif par une grammaire formelle. Pour dessiner des courbes, vous pourriez utiliser un outil de dessin modélisé par une position et un angle comme une "Tortue". Nous proposons d'implémenter la courbe de Gosper. Cette courbe est donnée par la grammaire suivante :

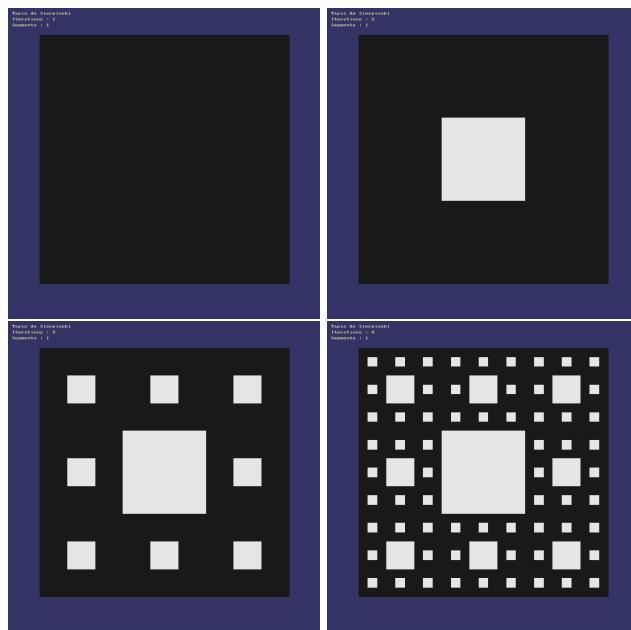
- Alphabet (commandes) :
 - A : dessin si degré 1, appel récursif sinon.
 - B : dessin si degré 1, appel récursif sinon.
 - $+$: tourner de 60° sens anti-horaire (vers la gauche).
 - $-$: tourner de 60° sens horaire (vers la droite).
- Axiome (règle initiale) : A .
- Règles (schémas récursifs) :
 - $A : A - B - -B + A + +AA + B -$
 - $B : +A - BB - -B - A + +A + B$

Le tracé en suivant ces règles peut donner le résultat suivant :

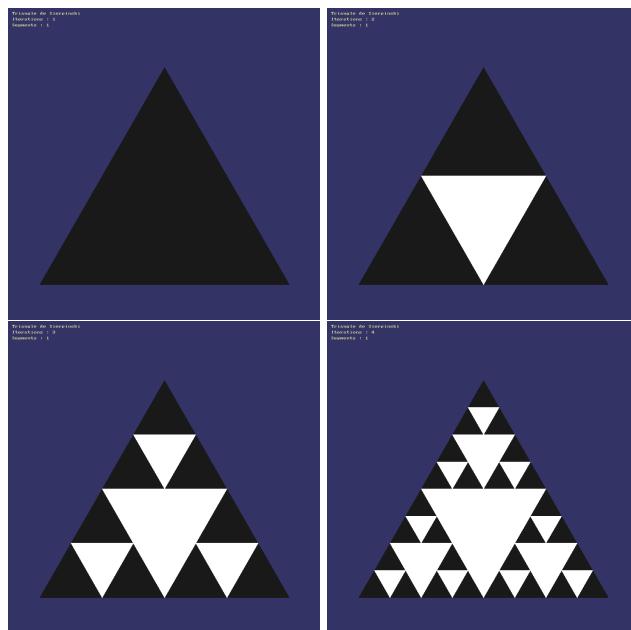


Les fonctions récursives peuvent aussi modéliser plus que des courbes. Ceci peut par exemple se faire sur des surfaces comme celles de Sierpiński :

- Tapis de Sierpiński :



- Triangle de Sierpiński :



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI graph";
const int functions_count = 9;

/* Position de la caméra et échelle de la vue : */
float cx, cy, cz;

int current_function = 0;
int iterations = 2;
int segments = 1;

void (*fonction(int id))(SDL_Surface * ecran, double sx, double sy,
→ double ex, double ey, int n, int r, int g, int b, int a);

const char * fonction_name(int id);

int ecran_depuis_camera_x(float x) {
    /* TODO */
    return x;
}

int ecran_depuis_camera_y(float y) {
    /* TODO */
    return y;
}

float ecran_depuis_camera_z(int z) {
    /* TODO */
    return z;
}

float camera_depuis_ecran_x(int x) {
    /* TODO */
    return x;
}
```

```

float camera_depuis_ecran_y(int y) {
    /* TODO */
    return y;
}

float camera_depuis_ecran_z(int z) {
    /* TODO */
    return z;
}

void affichage(SDL_Surface * ecran) {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));

    void (*line)(SDL_Surface * ecran, double sx, double sy, double ex,
    ↵ double ey, int n, int r, int g, int b, int a) =
    ↵ fonction(current_function);

    double scale = 10;
    if(segments <= 1) {
        line(ecran,
            ecran_depuis_camera_x(-scale), ecran_depuis_camera_y(0),
            ecran_depuis_camera_x(scale), ecran_depuis_camera_y(0),
            iterations,
            255, 255, 255, 255);
    } else {
        int i;
        for(i = 0; i < segments; ++i) {
            line(ecran,
                ecran_depuis_camera_x(scale * cos(i * 2 * M_PI / segments)),
                ↵ ecran_depuis_camera_y(scale * sin(i * 2 * M_PI / segments)),
                ecran_depuis_camera_x(scale * cos((i + 1) * 2 * M_PI /
                ↵ segments)), ecran_depuis_camera_y(scale * sin((i + 1) * 2 *
                ↵ M_PI / segments)),
                iterations,
                255, 255, 255, 255);
        }
    }

    char buffer[1024];
    stringRGBA(ecran, 10, 10, fonction_name(current_function), 204, 204,
    ↵ 153, 255);
    sprintf(buffer, "Iterations : %d", iterations);
    stringRGBA(ecran, 10, 25, buffer, 204, 204, 153, 255);
    sprintf(buffer, "Segments : %d", segments);
}

```

```
    stringRGB(ecran, 10, 40, buffer, 204, 204, 153, 255);
}

void flocon_koch(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : flocon de Koch */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_koch(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : courbe de Koch */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_koch_alt(SDL_Surface * ecran, double sx, double sy, double
→ ex, double ey, int n, int r, int g, int b, int a) {
    /* TODO : courbe de Koch version alternée */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_cesaro(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : fractale de Cesaro */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_dragon(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : courbe du Dragon */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_gosper(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : courbe de Gosper */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}

void courbe_hilbert(SDL_Surface * ecran, double sx, double sy, double ex,
→ double ey, int n, int r, int g, int b, int a) {
    /* TODO : courbe de Hilbert */
    lineRGB(ecran, sx, sy, ex, ey, r, g, b, a);
}
```

```

void tapis_sierpinski(SDL_Surface * ecran, double sx, double sy, double
← ex, double ey, int n, int r, int g, int b, int a) {
    int nx = (ey - sy);
    int ny = -(ex - sx);
    Sint16 xs[] = {
        sx,
        sx + nx,
        ex + nx,
        ex
    };
    Sint16 ys[] = {
        sy,
        sy + ny,
        ey + ny,
        ey
    };
    filledPolygonRGBA(ecran, xs, ys, 4, r / 10, g / 10, b / 10, a);
    /* TODO : tapis de Spierpinski */
}

void triangle_sierpinski(SDL_Surface * ecran, double sx, double sy,
← double ex, double ey, int n, int r, int g, int b, int a) {
    int nx = (ey - sy);
    int ny = -(ex - sx);
    Sint16 xs[] = {
        sx,
        0.5 * (sx + ex) + 0.87 * nx,
        ex
    };
    Sint16 ys[] = {
        sy,
        0.5 * (sy + ey) + 0.87 * ny,
        ey
    };
    filledPolygonRGBA(ecran, xs, ys, 3, r / 10, g / 10, b / 10, a);
    /* TODO : triangle de Spierpinski */
}

const char * fonction_name(int id) {
    switch(id) {
        case 1 : return "Courbe de Koch";
        case 2 : return "Fractale Cesaro";
        case 3 : return "Courbe de Koch alternee";
        case 4 : return "Courbe du Dragon";
        case 5 : return "Courbe de Gosper";
    }
}

```

```
    case 6 : return "Courbe de Hilbert";
    case 7 : return "Tapis de Sierpinski";
    case 8 : return "Triangle de Sierpinski";
    default : return "Flocon de Koch";
}
}

void (*fonction(int id))(SDL_Surface * ecran, double sx, double sy,
↪ double ex, double ey, int n, int r, int g, int b, int a) {
switch(id) {
    case 1 : return courbe_koch;
    case 2 : return courbe_cesaro;
    case 3 : return courbe_koch_alt;
    case 4 : return courbe.dragon;
    case 5 : return courbe_gosper;
    case 6 : return courbe_hilbert;
    case 7 : return tapis_sierpinski;
    case 8 : return triangle_sierpinski;
    default : return flocon_koch;
}
}

int main() {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
↪ SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    /* Placement du joueur au centre de la fenêtre : */

    cx = 0;
    cy = 0;
    cz = 10;

    int active = 1;
```

```
SDL_Event event;
int grab = 0;
int refresh = 1;

while(active) {

    if(refresh) {
        affichage(ecran);
        SDL_Flip(ecran);
        refresh = 0;
    }

    while(SDL_PollEvent(&event)) {

        switch(event.type) {
            /* Utilisateur clique sur la croix de la fenêtre : */
            case SDL_QUIT : {
                active = 0;
            } break;

            /* Utilisateur enfonce une touche du clavier : */
            case SDL_KEYDOWN : {
                switch(event.key.keysym.sym) {
                    /* Touche Echap : */
                    case SDLK_ESCAPE : {
                        active = 0;
                    } break;
                }
            } break;

            case SDL_KEYUP : {
                switch(event.key.keysym.sym) {
                    case SDLK_UP : {
                        current_function = (current_function + 1) %
                            functions_count;
                        refresh = 1;
                    } break;

                    case SDLK_DOWN : {
                        current_function = (current_function + functions_count - 1)
                            % functions_count;
                        refresh = 1;
                    } break;
                }
            } break;

            case SDLK_RIGHT : {
```

```
    iterations++;
    refresh = 1;
} break;

case SDLK_LEFT : {
    iterations--;
    refresh = 1;
} break;

case SDLK_a : {
    segments++;
    refresh = 1;
} break;

case SDLK_q : {
    segments--;
    refresh = 1;
} break;
}

} break;

case SDL_MOUSEBUTTONDOWN : {
switch(event.button.button) {
    case SDL_BUTTON_WHEELUP : {
        cz *= 0.8;
        if(cz < 1e-9) {
            cz = 1e-9;
        }
        refresh = 1;
    } break;
}

case SDL_BUTTON_WHEELDOWN : {
        cz /= 0.8;
        if(cz > 1e9) {
            cz = 1e9;
        }
        refresh = 1;
    } break;
}

case SDL_BUTTON_LEFT : {
    grab = 1;
} break;
}

} break;
```

```
case SDL_MOUSEBUTTONDOWN : {
    switch(event.button.button) {
        case SDL_BUTTON_LEFT : {
            grab = 0;
            } break;
        }
    } break;

case SDL_MOUSEMOTION : {
    if(grab) {
        cx += camera_depuis_ecran_z(-event.motion.xrel);
        cy += camera_depuis_ecran_z(-event.motion.yrel);
        refresh = 1;
        }
    } break;
}

SDL_Delay(1000 / 60);
}

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

7 Tableaux

7.1 Tableau à une dimension

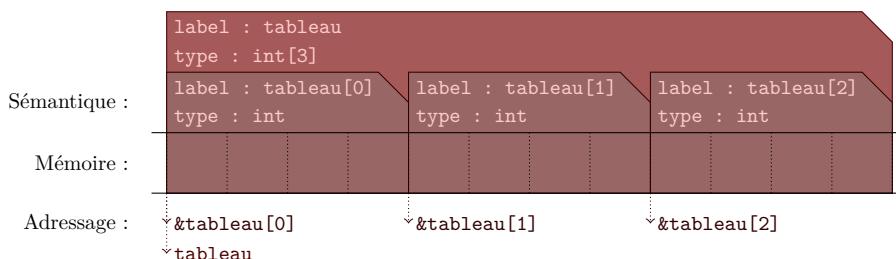
Nous avons vu comment gérer des variables, mais on peut vouloir gérer une liste de mêmes valeurs, comme les températures des derniers mois ou encore une table des plus hauts scores obtenus par des joueurs. Il ne semble pas acceptable de déclarer une variable à la main surtout si on maintient un tableau de score des 100 derniers joueurs par exemple. Pour faire ceci, nous pouvons construire des tableaux.

Un tableau se déclare comme une variable avec des crochets `[]`. On peut lors de la déclaration du tableau indiquer les valeurs qui le composent par défaut et aussi indiquer à l'avance une taille souhaitée si nous n'avons pas ces informations à la déclaration. L'accès à un élément du tableau par son indice se fait ensuite par la syntaxe `tableau[indice]` et peut être géré comme une variable.

```
type nomTableau[] = {valeur1, valeur2, ..., valeurN};  
type nomTableau[TAILLE];  
nomTableau[indice]; /* Appel élément d'indice 'indice' */
```

Ceci se transposerait par exemple comme il suit pour construire un tableau de trois entiers.

```
int tableau[] = {1, 2, 3};  
int tableau[3];  
tableau[0] = 1; /* affectation première valeur */
```



À bien noter que les indices des tableaux commencent à 0.

```
int liste[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, -1};
int i;
for(i = 0; liste[i] >= 0; ++i) {
    if(i) {
        printf(", ");
    }
    printf("%d", liste[i]);
}
printf("\n");
```

En sortie :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

Plutôt que de saisir ces valeurs à la main, il est possible de souhaiter construire les valeurs de la liste vue précédemment (Cette séquence est la suite de Fibonacci) :

```
const int taille_liste = 16;
int liste[taille_liste];
int i;
for(i = 0; i < taille_liste; ++i) {
    if(i < 2) {
        liste[i] = i;
    } else {
        liste[i] = liste[i - 1] + liste[i - 2];
    }
}
for(i = 0; i < taille_liste; ++i) {
    if(i) {
        printf(", ");
    }
    printf("%d", liste[i]);
}
printf("\n");
```

En sortie :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610
```

À noter qu'en langage C, lorsqu'on fabrique un tableau, il correspond uniquement au fait de demander un nombre d'éléments successif en mémoire et l'appel au nom du tableau donnera l'adresse du premier élément :

```
int tableau[] = {1, 2, 3};  
printf("adresse de tableau : %p\n", tableau);  
printf("adresse de tableau[0] : %p\n", &tableau[0]);  
printf("adresse de tableau[1] : %p\n", &tableau[1]);  
printf("adresse de tableau[2] : %p\n", &tableau[2]);
```

En sortie :

```
adresse de tableau : 0x7ffcff40ff4c  
adresse de tableau[0] : 0x7ffcff40ff4c  
adresse de tableau[1] : 0x7ffcff40ff50  
adresse de tableau[2] : 0x7ffcff40ff54
```

En effet, l'adresse du premier élément correspond à au nom du tableau et ici, nos éléments ont des adresses espacées de la taille d'un `int` soit 4 octets. Un tableau est un peu comme une rue de maisons où les maisons se trouvent à adresses successives et sont séparées par l'espace qu'occupe une maison.

À noter que l'opérateur `sizeof` permet de déterminer la taille en octets d'un type ou d'un élément.

```
printf("sizeof(char) : %lu\n", sizeof(char));  
printf("sizeof(short) : %lu\n", sizeof(short));  
printf("sizeof(int) : %lu\n", sizeof(int));  
printf("sizeof(long) : %lu\n", sizeof(long));
```

En sortie :

```
sizeof(char) : 1  
sizeof(short) : 2  
sizeof(int) : 4  
sizeof(long) : 8
```

Il est possible de passer un tableau à une fonction sans préciser sa taille :

```

void afficher_liste(int liste[]) {
    int i;
    for(i = 0; liste[i] >= 0; ++i) {
        if(i) {
            printf(", ");
        }
        printf("%d", liste[i]);
    }
    printf("\n");
}

int main() {
    int liste[12] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, -1};
    afficher_liste(liste);
    exit(EXIT_SUCCESS);
}

```

En sortie :

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

Cependant, bien qu'utiliser `sizeof` sur un tableau ou une variable pour connaître sa taille semble intéressant, il reste à utiliser avec parcimonie car ceci pourra être source d'erreur si utilisé dans une fonction. Préférez une connaissance du type et de ce que fait votre code. En effet, dans une fonction un tableau est considéré comme son adresse, c'est la taille de l'adresse qui sera considérée :

```

void afficher_taille_tableau(float tableau[]) {
    printf("depuis une fonction : %lu\n", sizeof(tableau));
}

int main() {
    float tableau[16];
    printf("depuis main : %lu\n", sizeof(tableau));
    afficher_taille_tableau(tableau);
    exit(EXIT_SUCCESS);
}

```

En sortie :

```
depuis main : 64
depuis une fonction : 8
```

7.2 Chaînes de caractères

Les tableaux ont aussi un intérêt particulier puisqu'ils permettent de construire des chaînes de caractères. En réalité ce sont des tableaux de caractères. À noter qu'une chaîne de caractères se termine par le symbole '\0' :

```
char tableau_de_char[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', ' ', 
    ~ '!', '\0'};
int i;
for(i = 0; tableau_de_char[i] != '\0'; ++i) {
    putchar(tableau_de_char[i]);
}
putchar('\n');
```

En sortie :

```
Bonjour !
```

En réalité, le langage C permet de déclarer une chaîne de caractères depuis une chaîne constante donnée entre double guillemets et afficher une chaîne de caractères de manière moins fastidieuse depuis le format %s :

```
char texte[] = "Bonjour !";
printf("%s\n", texte);
```

En sortie :

```
Bonjour !
```

Une chaîne de caractère peut aussi être lue depuis l'entrée standard de la console avec scanf et le format %s :

```
char nom[64];
printf("Quel est votre nom ? ");
```

```
scanf("%s", nom);
printf("Bonjour %s !\n", nom);
```

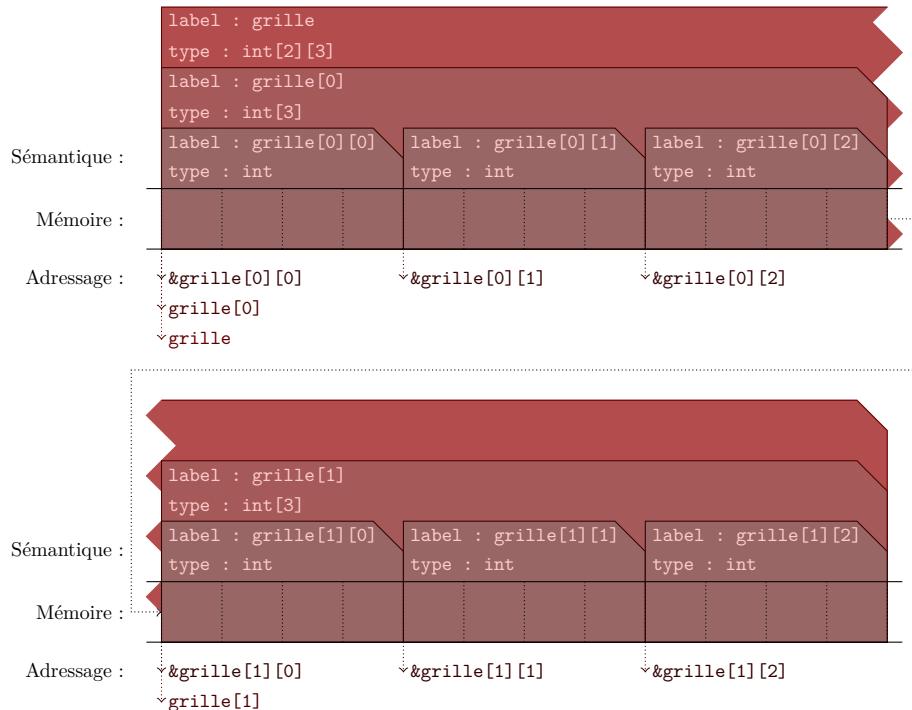
En sortie :

```
Quel est votre nom ? M.Trancho
Bonjour M.Trancho !
```

À noter qu'un bibliothèque standard, `string.h`, est dédiée à la manipulation de chaînes de caractères. Recoder ses fonctions et découvrir ses fonctionnalités fera partie des exercices.

7.3 Tableau à plusieurs dimensions

Imaginons maintenant que nous aimerais définir un terrain où vivrait notre personnage, ceci peut se faire avec un tableau, mais il existe une alternative : les tableaux à plusieurs dimensions.



Par exemple un tableau à deux dimensions nous permettrait d'avoir la possibilité de renseigner deux indices au lieu de un ce qui convient à la définition d'une grille. Ceci se déclare comme un tableau, mais il faudra ajouter des crochets pour chaque dimension supplémentaire :

```
int grille[2][3];
/* 2 lignes */
/* 3 colonnes */

char terrain[5][6] = {
    "#.###",
    "#.#.#",
    "#...#",
    "#.#.#",
    "###.#"
};

int ligne, colonne;
for(ligne = 0; ligne < 5; ++ligne) {
    for(colonne = 0; colonne < 6; ++colonne) {
        putchar(terrain[ligne][colonne]);
    }
    putchar('\n');
}
```

En sortie :

```
#.###
#.#
#...
#.#
###.
```

Cependant, lorsqu'un tableau à plusieurs dimensions est passé à une fonction, seule sa première dimension peut être laissé sans valeur fixe, les autres doivent être renseignées.

```
void afficher_terrain(char terrain[][6], int hauteur) {
    int ligne, colonne;
    for(ligne = 0; ligne < hauteur; ++ligne) {
        for(colonne = 0; colonne < 6; ++colonne) {
```

```

        putchar(terrain[ligne][colonne]);
    }
    putchar('\n');
}
}

int main() {
    char terrain[5][6] = {
        "#.###",
        "#.#.#",
        "#...#",
        "#.#.#",
        "###.#"
    };
    afficher_terrain(terrain, 5);
    exit(EXIT_SUCCESS);
}

```

En sortie :

```
#.###
#.#
#...
#.#
###.
```

L'astuce pour passer un tableau à plusieurs dimensions à une fonction sans dimensions figées peut être de placer la récupération des dimensions avant le tableau dans la liste d'arguments :

```

void afficher_terrain(int hauteur, int largeur, char
    ↳ terrain[hauteur][largeur]) {
    int ligne, colonne;
    for(ligne = 0; ligne < hauteur; ++ligne) {
        for(colonne = 0; colonne < largeur - 1; ++colonne) {
            putchar(terrain[ligne][colonne]);
        }
        putchar('\n');
    }
}

```

```
int main() {
    char terrain[5][7] = {
        "#.#####",
        "#.#..#",
        "#...##",
        "#.#..#",
        "####.#"
    };
    afficher_terrain(5, 7, terrain);
    exit(EXIT_SUCCESS);
}
```

En sortie :

```
#.#####
#.#
#...
#.#..
####.#
```

7.4 Résumé

Un tableau à une dimension peut se déclarer de la manière suivante :

```
[TYPE] [NOM_TABLEAU][] = {[VALEUR 1], ..., [VALEUR N]};  
[TYPE] [NOM_TABLEAU][[NOMBRE VALEURS]];
```

Chaque élément d'un tableau est une variable et l'accès à l'élément d'indice `indice` se fait comme il suit :

```
tableau[indice];
```

La valeur que représente le nom du tableau est l'adresse de son premier élément. Les éléments d'un tableau sont alignés en mémoire et leurs adresses sont séparées par la taille du type du tableau.

L'opérateur `sizeof` permet de déterminer l'espace qu'occupe un type en mémoire.

Une chaîne de caractères est un tableau de `char` et peut être définie par une chaîne constante :

```
char texte[] = "Texte";
```

Une chaîne de caractères a pour dernier élément le caractère nul '`\0`'.

Une chaîne de caractères se gère en entrée-sortie par le format `%s` :

```
printf("%s\n", texte);  
scanf("%s", texte);
```

Un tableau à plusieurs dimensions peut se déclarer de la manière suivante :

```
[TYPE] [NOM_TABLEAU][[DIMENSION 1]]...[[DIMENSION N]];
```

Son accès se fait de la manière suivante :

```
tableau[indice_1][indice_2]...[indice_n];
```

7.5 Entraînement

Exercice 59 (** Creation et affectation).

Completer le code suivant pour de sorte de :

- Deфинir l'argument de la fonction `afficher_tableau`.
- Ecrire le code de la fonction `afficher_tableau`.
- Ecrire la deфинition et affectation du tableau.
- Changer une valeur du tableau.

```
#include <stdio.h>
#include <stdlib.h>

void afficher_tableau(/* TODO : passer un tableau */) {
    /* TODO : afficher les valeurs positives sous la forme : */
    /* [1, 2, 3, 4, 5] */
}

int main() {
    /* TODO : deфинir un tableau d'entiers avec les valeurs */
    /* suivantes : 1, 2, 3, 4, 5 et -1 */
    int indice;
    int valeur;
    afficher_tableau(tableau);
    printf("Quel indice affecter ? ");
    scanf("%d", &indice);
    printf("Pour quelle valeur ? ");
    scanf("%d", &valeur);
    /* TODO : affecter valeur a l'indice "indice" du tableau */
    afficher_tableau(tableau);
    exit(EXIT_SUCCESS);
}
```

Exercice 60 (★★ Lire valeurs).

Compléter le code suivant pour de sorte de :

- D'afficher un tableau connaissant sa taille.
- Définir un tableau d'une taille donnée.
- Lire des valeurs et les ranger dans le tableau.

```
#include <stdio.h>
#include <stdlib.h>

void afficher_tableau(int tableau[], int taille) {
    /* TODO : afficher les valeurs du tableau */
}

int main() {
    const int taille = 5;
    /* TODO : Définir un tableau d'entiers de taille "taille" */
    int i;
    printf("Saisir %d valeurs : ", taille);
    /* TODO : Lire "taille" valeurs */
    afficher_tableau(tableau, taille);
    exit(EXIT_SUCCESS);
}
```

Exercice 61 (★★ Statistiques sur un tableau).

Complétez le code ci-dessous pour qu'il donne la sortie suivante :

```
minimum : valeurs[6] = 0
maximum : valeurs[1] = 9
moyenne : 4.5
```

```
#include <stdio.h>
#include <stdlib.h>

#define TAILLE 10
/* min_index renvoie l'indice du plus petit élément de values */
int min_index(int values[], int taille);
/* min_value renvoie le plus petit élément de values */
int min_value(int values[], int taille);
/* max_index renvoie l'indice du plus grand élément de values */
int max_index(int values[], int taille);
/* max_value renvoie le plus grand élément de values */
int max_value(int values[], int taille);
/* moyenne renvoie la moyenne des éléments de values */
float moyenne(int values[], int taille);

int main() {
    int valeurs[TAILLE] = {5, 9, 1, 4, 8, 3, 0, 6, 2, 7};
    printf("minimum : valeurs[%d] = %d\n", min_index(valeurs,
        → TAILLE), min_value(valeurs, TAILLE));
    printf("maximum : valeurs[%d] = %d\n", max_index(valeurs,
        → TAILLE), max_value(valeurs, TAILLE));
    printf("moyenne : %g\n", moyenne(valeurs, TAILLE));
    exit(EXIT_SUCCESS);
}
```

Exercice 62 (★ Coder les fonction de string.h).

Bob aimerait voir si vous savez recoder les fonctions classiques de la bibliothèque `string.h` et vous met au défit d'en recoder quelques unes juste pour voir. Il vous indique rapidement ce qu'elles font :

1. `strlen` donne la taille de la chaîne de caractères.
2. `strcpy` copie une chaîne de caractères dans une autre.
3. `strcat` ajoute une chaîne de caractères à une autre.
4. `strcmp` renvoie 0 si les chaînes sont identiques, une valeur négative si la première précède la seconde dans l'ordre lexicographique et une valeur positive si c'est l'inverse.

À vous de les coder en version `esgi_` et vérifier que le comportement est semblable à celles de la bibliothèque standard :

```
#include <stdio.h>
#include <stdlib.h>

int esgi_strlen(const char texte[]);
void esgi strcpy(char destination[], const char source[]);
void esgi strcat(char destination[], const char source[]);
int esgi strcmp(const char first[], const char second[]);

int main() {
    char texte[] = "Welcome to ESGI !";
    char hello[] = "Hello";
    char copie[50];
    printf("esgi_strlen(\"%s\") = %d\n", texte, esgi_strlen(texte));
    esgi strcpy(copie, "Eleve, ");
    printf("copie = \"%s\"\n", copie);
    esgi strcat(copie, texte);
    printf("copie = \"%s\"\n", copie);
    printf("esgi strcmp(\"Hello\", \"Hello\") = %d = 0\n",
        esgi strcmp(hello, "Hello"));
    printf("esgi strcmp(\"Hello\", \"Bonjour\") = %d > 0\n",
        esgi strcmp(hello, "Bonjour"));
    printf("esgi strcmp(\"Hello\", \"Hell\") = %d > 0\n",
        esgi strcmp(hello, "Hell"));
    printf("esgi strcmp(\"Bonjour\", \"Hello\") = %d < 0\n",
        esgi strcmp("Bonjour", hello));
    exit(EXIT_SUCCESS);
}
```

Exercice 63 (★★★ Compter les occurrences de lettres).

Alice souhaite compter les occurrences de lettres (caractères alphabétiques) dans une chaîne de caractères.

Par exemple, elle aimerait un programme qui lui dise que "Alice" est composé de : un 'a', un 'c', un 'e', un 'l' et un 'i'. Elle trouve amusant que pour "Bob" (deux 'b' et un 'o') le 'b' soit présent deux fois.

Elle vous indique qu'elle utiliserait votre programme avec au maximum une chaîne de caractères de 1 000 lettres et souhaiterait lister les occurrences des lettres dans l'ordre alphabétique. Votre programme pourrait donner les sorties suivantes :

```
Entrez du texte : Alice
Texte : "Alice"
'a' : 1
'c' : 1
'e' : 1
'i' : 1
'l' : 1
```

```
Entrez du texte : Bob
Texte : "Bob"
'b' : 2
'o' : 1
```

```
Entrez du texte : Exemple de texte.
Texte : "Exemple de texte."
'd' : 1
'e' : 6
'l' : 1
'm' : 1
'p' : 1
't' : 2
'x' : 2
```

Exercice 64 (★★★ Décodage Vigenère).

Oscar a trouvé sur internet une méthode connue pour envoyer des messages secrets : le Chiffre de **Vigenère**. Il a vanté les mérites de l'**ESGI**, a gribouillé "Fgtrsmx mxmjqefz li d'KAKA, swm u'kax Gykej, zc tjkniika pw rirygoi U uc Tqzpsf?" sur un tableau et a disparu sans donner plus d'explications. Pourriez-vous mettre au point un programme pour décoder ce message et lui répondre ?

Exercice 65 (★★★ Gestion d'une liste d'entiers).

Proposez un code qui permet de gérer une liste d'entier et propose les opérations suivantes :

- Ajouter une valeur
- Supprimer la dernière valeur
- Rechercher une valeur
- Afficher la liste
- Afficher des statistiques sur la liste (voir sortie)
- Trier la liste

```
>>> help
[Info] : Action invalide.
Actions possibles :
- ajouter [VALEUR]
- supprimer
- rechercher [VALEUR]
- afficher
- statistiques
- trier
- quitter
>>> afficher
Liste : []
>>> ajouter 42
>>> afficher
Liste : [42]
>>> rechercher 42
42 trouvé à l'indice 0.
>>> rechercher 1337
... (voir page suivante)
```

```
... (voir page précédente)
>>> ajouter 1337
>>> ajouter 1235
>>> ajouter 1
>>> ajouter 1000
>>> ajouter 7
>>> afficher
Liste : [42, 1337, 1235, 1, 1000, 7]
>>> statistiques
Statistiques :
- moyenne :      603.667
- ecart type :   595.557
- minimum :       1
- 1er quartile : 24.5
- mediane :      1000
- 3eme quartile : 1286
- maximum :      1337
>>> afficher
Liste : [42, 1337, 1235, 1, 1000, 7]
>>> trier
>>> afficher
Liste : [1, 7, 42, 1000, 1235, 1337]
>>> supprimer
[Info] : dernière valeur de la liste : 1337
>>> quitter
```

Exercice 66 (★★★ Dichotomie dans un tableau).

Bob souhaite maintenir une liste d'entiers saisis par un utilisateur. Cependant, il ne veut pas faire une gestion naïve de cette liste. Il a découvert qu'en maintenant sa liste d'entiers triée, il pouvait optimiser la recherche d'un élément par la méthode de la dichotomie.

Algorithm 3: Recherche dichotomique dans un tableau trié.

```

input : Liste liste : liste d'éléments.
input : Entier valeur : valeur recherchée.
output: Entier milieu : indice de la valeur dans la liste (négatif si non trouvé).

1 début  $\leftarrow$  0 ;
2 fin  $\leftarrow$  longueur (liste) ;
3 Tant que  $\text{fin} - \text{début} > 0$  faire
4   | milieu  $\leftarrow$   $\lfloor \frac{\text{début} + \text{fin}}{2} \rfloor$  ;
5   | Si liste[milieu] < valeur Alors
6   |   | début  $\leftarrow$  milieu + 1 ;
7   | Fin
8   | Sinon si liste[milieu] > valeur Alors
9   |   | fin  $\leftarrow$  milieu ;
10  | Fin
11 | Sinon
12 |   | Renvoyer milieu ;
13 | Fin
14 Fin
15 Si liste[début] = valeur Alors
16   |   Renvoyer début ;
17 Fin
18 /* V n'a pas été trouvée dans la liste : */          */
19 Renvoyer -1 ;

```

Bob aimeraient un programme un peu interactif pour ajouter, rechercher et afficher sa liste. Il vous indique qu'il resterait raisonnable et ajouterait au maximum 1 000 éléments. Les commandes pourraient être modélisées par les symboles suivants :

- '+' : ajouter un élément.
- '=' : rechercher un élément.
- ":" : afficher la liste.
- 'q' : arrêter le programme.

```
>>> .
[]
>>> + 4
>>> .
[4]
>>> + 6
>>> + 2
>>> .
[2, 4, 6]
>>> = 4
4 existe dans la liste.
>>> = 2
2 existe dans la liste.
>>> = 6
6 existe dans la liste.
>>> = 1
1 non trouvé dans la liste.
>>> = 3
3 non trouvé dans la liste.
>>> = 5
5 non trouvé dans la liste.
>>> = 7
7 non trouvé dans la liste.
>>> q
```

Exercice 67 (★★★ Suite de Fibonacci).

Alice, Bob et Charlie ont entendu parler de la suite de Fibonacci en cours d'algorithmique. Cette suite est donnée pour tout entier naturel n par la définition suivante :

$$\mathcal{F}_n = \begin{cases} 0 & Si\ n = 0 \\ 1 & Si\ n = 1 \\ \mathcal{F}_{n-1} + \mathcal{F}_{n-2} & Sinon \end{cases}$$

- Alice, propose d'implémenter la suite de Fibonacci depuis sa définition, c'est-à-dire par une fonction récursive.
- Bob pense que ce n'est pas la meilleure manière de la coder et qu'une simple boucle suffit pour passer d'un élément au suivant. Il suffirait de sauvegarder les valeurs aux indices précédents pour calculer le suivant.
- Charlie dit que c'est juste une suite récurrente linéaire et qu'il suffit de mettre la matrice $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ à la bonne puissance puisque

$$\begin{pmatrix} \mathcal{F}_{n+1} \\ \mathcal{F}_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} \mathcal{F}_1 \\ \mathcal{F}_0 \end{pmatrix}.$$

Il ajoute qu'avec l'exponentiation rapide, il battrait la méthode de Bob si on suppose vouloir prendre un indice n assez grand.

Implémentez les différentes méthodes et proposez des exemples pertinents permettant de mettre en évidence ou non les différences d'efficacités de chacune. Justifiez de ce qui à votre avis provoque ces différences de performances.

Notez que Charlie a eu l'amabilité de nous rappeler pour le cas d'usage la multiplication matrice - vecteur :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

et matrice - matrice :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x & z \\ y & w \end{pmatrix} = \begin{pmatrix} ax + by & az + bw \\ cx + dy & cz + dw \end{pmatrix}$$

De plus, l'exponentiation rapide dont il parle suivrait l'algorithme suivant :

Algorithm 4: Exponentiation rapide

```
input : Entier valeur
input : Entier exposant
output: Entier resultat

1 resultat ← 1 ;
2 while exposant > 0 do
3   | if exposant modulo 2 = 1 then
4   |   | resultat ← resultat × valeur ;
5   | end
6   | valeur ← valeur × valeur ;
7   | exposant ← exposant / 2;
8 end
```

Exercice 68 (★★★ Chiffrement par permutation d'alphabet).

Oscar a constaté que son chiffrement n'était plus assez haut niveau pour vous. Il a donc décidé de la changer pour un chiffrement par permutation d'alphabet. L'idée : la clé est donnée par un mélange des lettres de l'alphabet. Nous regardons dans la clé la lettre donnée par l'indice correspondant au décalage de la lettre à coder dans l'alphabet :

Alphabet :	a b c d e f g h i j k l m n o p q r s t u v w x y z
	
Clé :	y b P w g h j z t q l a k m i o s v x u f c d r e n

Ceci fait par exemple, que la chaîne de caractères "Langage C" serait codée "Aymjyjg P". Pour décoder une telle chaîne, il faudrait par exemple déduire l'alphabet inverse et l'utiliser comme clé sur le texte codé. C'est-à-dire que ceci pourrait par exemple consister à réordonner les lettres de la clé pour obtenir l'association inverse :

Clé :	a b c d e f g h i j k l m n o p q r s t u v w x y z
	
Alphabet :	l b v w y u e f o g m k n z p c j x q i t r d s a h

Oscar aimerait que vous proposiez des arguments à passer à votre programme en ligne de commande :

- "alpha" : génère une permutation d'alphabet et son alphabet inverse.

```
./alphacode alpha
alpha : "ibnueqjfsvacyrkwmgltpodzhx"
ialpha : "kblwehryagosqcvufnitdjpzmx"
```

- "code" : propose de coder un texte depuis un alphabet donné par l'utilisateur ou proposé automatiquement sinon.

```
./alphacode code "ybpwghjztqlakmiosvxufcdren"
alpha : "ybpwghjztqlakmiosvxufcdren"
ialpha : "lbvwyuefogmknzpcjxqitrdssah"
Saisir une ligne : Langage C
Texte      : "Langage C"
Texte codé  : "Aymjyjg P"
```

- "decode" : proposer de décoder un texte depuis un alphabet donné par l'utilisateur.

```
./alphacode decode "ybpwghjztqlakmiosvxufcdren"  
alpha : "ybpwghjztqlakmiosvxufcdren"  
ialpha : "lbvwyuefogmknzpcjxqitrdsa"  
Saisir une ligne : Aymjyjg P  
Texte : "Aymjyjg P"  
Texte décodé : "Langage C"
```

Votre programme pourrait donner la sortie suivante :

```
alpha : "caqvtkfnuwghlrdispzyeomxjb"  
ialpha : "bzaougklyfmwhvrcnqeidjxts"  
Saisir une ligne : Exemple de texte super super safe !  
Texte : "Exemple de texte super super safe !"  
Texte codé : "Txtliht vt ytxyt zeitp zeitp zckt !"
```

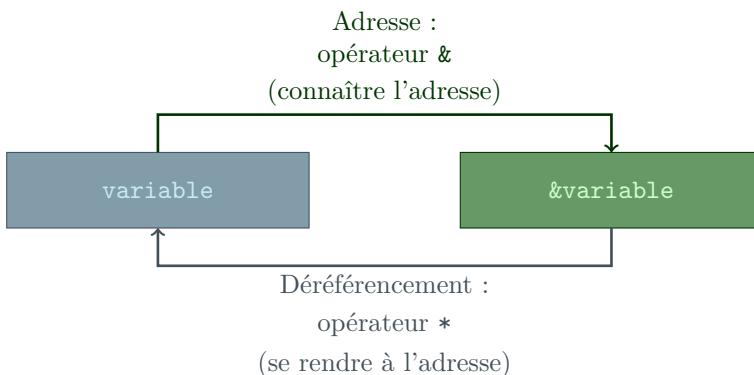
Oscar étant joueur, il vous a laissé un message avant de disparaître. Il a dit que vous comprendrez ce qu'il veut dire : "Rvk, cvxox j'tfayf qf hvk qfnx! U'mx cvjvkbmxyhfkb xkwoyxak bfgbf mwwfs jvkp, txwbvxyf qf cvaw nmoxjxbfy jfw evwwxrxjxbfw qf cfkxy m rvab qf of otxnnnyhfkb! Evay nmxyf cmjvxy cvbyf yfawwxbf, cvaw k'mayfs da'm qxyf daf jm yfevkwf fw!!! Kvk!".

8 Pointeurs

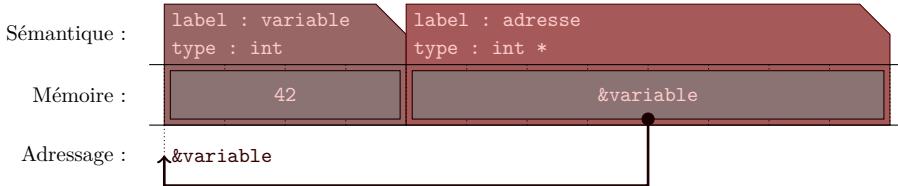
8.1 Un type d'adresse

Nous savons que chaque variable a une adresse, mais si nous voulons sauvegarder une adresse ou tenir un carnet d'adresse, il serait possible de le sauvegarder dans un `long` bien qu'à la fin nous ne saurions plus quel est le type de la variable représentée. Pour ceci nous avons un outil nommé **pointeur**. Un pointeur c'est une variable typée qui permet de représenter l'adresse d'une autre variable typée ou d'un emplacement mémoire.

Par exemple, si nous avions un type `Maison`, nous pourrions vouloir référencer l'adresse de cette `Maison` par un type `Adresse d'une Maison`. Ce type `Adresse d'une Maison` a pour syntaxe le type référencé suivi d'une étoile `*`. Autrement dit, l'adresse d'une `Maison` peut se sauvegarder dans une variable de type `Maison *`. Le type `Maison` est une illustration qui peut être remplacée par le type connus.



Cependant, si nous n'avons que l'adresse `Maison * adresse` d'une variable `Maison house`, nous ne pourrons pas en faire grand chose car `adresse` correspond à `&house`, ce qui nous intéresse c'est la variable elle-même (la `Maison house`). Pour aller à la `Maison house`, nous devons déréférencer la adresse, ce qui se fait en ajoutant une étoile `*` devant l'adresse : `*adresse` correspond à `house`.



```
int variable = 42;
int * adresse = &variable;
printf("Maison : variable : %d\n", variable);
printf("Adresse : &variable : %p\n", &variable);
printf("Adresse : adresse : %p\n", adresse);
printf("Maison : *adresse : %d\n", *adresse);
printf("Maison : *&variable : %d\n", *(&variable));
```

En sortie :

```
Maison : variable : 42
Adresse : &variable : 0x7fff76c890b4
Adresse : adresse : 0x7fff76c890b4
Maison : *adresse : 42
Maison : *&variable : 42
```

Exercice 69 (★★★ Fonction pour échanger des valeurs).

Un camarade fictif est lassé de déclarer une variable temporaire pour chaque échange de valeurs et veut coder une fonction qui le fasse. Cependant son code ne fonctionne pas. Sans changer la logique du code, proposer une solution pour le faire fonctionner :

```
void echanger(int first, int second) {
    int temporaire;
    temporaire = first;
    first = second;
    second = temporaire;
}

int main() {
```

```
1 int first = 42, second = 1337;
2 printf("first = %d, second = %d\n", first, second);
3 echanger(first, second);
4 printf("first = %d, second = %d\n", first, second);
5 exit(EXIT_SUCCESS);
6 }
```

Un pointeur est intéressant pour deux cas de figures :

1. Modifier la valeur d'une variable dans une fonction depuis son adresse.
2. Ne pas recopier tout un élément en le passant à une fonction par exemple, mais uniquement son adresse (tableaux).

8.2 Arithmétique des pointeurs

Nous avions vu précédemment qu'un tableau correspond en réalité à l'adresse de sa première valeur. En réalité, la manipulation par le nom du tableau par un pointeur sur le tableau est équivalente :

```
1 void afficher_tableau(int * tableau) {
2     int i;
3     for(i = 0; tableau[i] > 0; ++i) {
4         if(i) {
5             printf(" , ");
6         }
7         printf("%d", tableau[i]);
8     }
9     printf("\n");
10 }
11
12 int main() {
13     int tableau[] = {1, 2, 3, -1};
14     int * pointeur = tableau;
15     printf("tableau : %p\n", tableau);
16     printf("pointeur : %p\n", pointeur);
17     afficher_tableau(tableau);
18     pointeur[1] = 42;
19     afficher_tableau(tableau);
20     exit(EXIT_SUCCESS);
21 }
```

En sortie :

```
tableau : 0x7ffd298aeee40
pointeur : 0x7ffd298aeee40
1, 2, 3
1, 42, 3
```

Notons que dans ce code, nous récupérons le tableau comme un pointeur sur un tableau et que nous pouvons l'utiliser comme un tableau pour imprimer ses éléments. De même, à la ligne 18, nous utilisons le pointeur comme un tableau pour modifier l'élément d'indice 1.

Nous avions vu que pour un tableau, les adresses de chaque élément se suivaient en mémoire séparées par la taille qu'occupe le type en mémoire. En réalité, il est possible de jouer avec les adresses avec ce que l'on nomme **l'arithmétique des pointeur** : c'est-à-dire que si on ajoute 1 à l'adresse d'un élément d'un tableau ou un pointeur, on a l'adresse de l'élément suivant :

```
int tableau[] = {1, 2, 3, -1};
int * pointeur = tableau;
printf("tableau : %p\n", tableau);
printf("&tableau[1] : %p\n", &tableau[1]);
printf("&pointeur[1] : %p\n", &pointeur[1]);
printf("(tableau + 1) : %p\n", tableau + 1);
printf("(pointeur + 1) : %p\n", pointeur + 1);
```

En sortie :

```
tableau : 0x7fff25428330
&tableau[1] : 0x7fff25428334
&pointeur[1] : 0x7fff25428334
(tableau + 1) : 0x7fff25428334
(pointeur + 1) : 0x7fff25428334
```

Pour aller plus loin avec cette mécanique, nous pouvons déréférencer les adresses des différents éléments et obtenir leurs valeurs ce qui est équivalent à l'accès entre crochets (c'est ce que fait la machine) :

```
int tableau[] = {1, 2, 3, -1};
int * pointeur = tableau;
```

```
printf("&pointeur[1] : %p\n", &pointeur[1]);
printf("(pointeur + 1) : %p\n", pointeur + 1);
printf("pointeur[1] : %d\n", pointeur[1]);
printf("* (pointeur + 1) : %d\n", *(pointeur + 1));
printf("*(1 + pointeur) : %d\n", *(1 + pointeur));
printf("1 [pointeur] : %d\n", 1 [pointeur]);
```

En sortie :

```
&pointeur[1] : 0x7ffdab94b194
(pointeur + 1) : 0x7ffdab94b194
pointeur[1] : 2
*(pointeur + 1) : 2
*(1 + pointeur) : 2
1 [pointeur] : 2
```

À noter que les deux dernières lignes sont plus une démonstration qu'un concept à utiliser dans votre code : pour montrer le pointeur et l'indice sont interchangeables et que la machine applique exactement cette opération de déréférencement pour la notation entre crochets.

À noter que ceci est compatible avec l'idée de modifier la valeur d'un pointeur pour itérer sur ses éléments comme pour une chaîne de caractères :

```
void afficher_chaine(char * chaine) {
    for(; *chaine != '\0'; ++chaine) {
        putchar(*chaine);
    }
    putchar('\n');
}

int main() {
    char texte[] = "Hello ESGI !";
    afficher_chaine(texte);
    exit(EXIT_SUCCESS);
}
```

```
Hello ESGI !
```

Par exemple, ceci permet d'utiliser l'opérateur `++` sur le pointeur pour passer d'une adresse à la suivante (`pointeur = pointeur + 1`) et obtenir chaque caractère, on déréférence l'adresse courante.

8.3 Allocation dynamique

Nous avons vu comment obtenir des tableaux, mais le problème principal est que leur taille reste en général figée à la compilation. Il est possible de réclamer un espace mémoire dont la taille est donnée à l'exécution par l'allocation dynamique.

De la même manière qu'un tableau est donné par l'adresse de son premier élément, il est possible de récupérer l'adresse du premier élément d'une plage réclamée par appel à la fonction `malloc`. On demande à `malloc` une plage mémoire d'une taille donnée en octets et `malloc` renvoie une plage valide si réussi et `NULL` sinon. Il faudra penser à libérer la mémoire en fin de programme ou lorsqu'il n'est plus intéressant de la garder en mémoire avec la fonction `free`. Une manière simple de faire appel à `malloc` est la suivante :

```
type * tableau = malloc(sizeof(type) * taille);
/* équivalent à type tableau[taille] avec taille une variable */
/* utiliser tableau */
free(tableau);
```

Une manière propre de gérer la mémoire peut se faire de la manière suivante :

```
type * tableau = NULL;
if((tableau = (type *)malloc(sizeof(type) * taille)) == NULL) {
    /* traitement à appliquer, sauvegarde si possible */
    exit(EXIT_FAILURE);
}
/* utiliser tableau */
free(tableau);
tableau = NULL;
```

Nous maintenons un pointeur qui ne pointe vers aucune adresse et aucun emplacement mémoire à `NULL`. Dans le cas où l'allocation par `malloc` échoue, c'est en réalité souvent critique et difficilement rattrapable, donc en général, il faudrait sauvegarder si possible le travail de l'utilisateur et terminer le programme. Une fois que la plage mémoire est libérée par `free`, le pointeur regarde une adresse devenue invalide, on repasse sa valeur à `NULL`.

Pour vérifier que le programme libère bien tout la mémoire qu'il a demandé, il est possible d'utiliser la commande `valgrind [LANCEMENT DU PROGRAMME]` pour le vérifier. Si toute la mémoire réclamée a bien été libérée, alors la commande doit terminer par une sortie semblable à la suivante :

```
==3396== HEAP SUMMARY:
==3396==     in use at exit: 0 bytes in 0 blocks
==3396==   total heap usage: 1 allocs, 1 frees, 168 bytes
→  allocated
==3396==
==3396== All heap blocks were freed -- no leaks are possible
```

Il existe aussi d'autres fonctions d'allocation qui effectuent des actions supplémentaires :

- `calloc` qui alloue une plage mémoire et assure tous les éléments sont initialisés à 0.
- `realloc` qui permet de changer la taille d'une plage mémoire déjà allouée.

Par exemple les instructions suivantes allouent une plage mémoire, puis agrandissent la plage mémoire en gardant les élément précédemment calculés :

```
int i;
int taille = 5;
int * tableau = NULL;
if((tableau = (int *)calloc(taille, sizeof(int))) == NULL) {
    exit(EXIT_FAILURE);
}
for(i = 0; i < taille; ++i) {
    tableau[i] = i;
}
for(i = 0; i < taille; ++i) {
    if(i) { printf(", "); }
    printf("%d", tableau[i]);
}
printf("\n");

taille = 10;
if((tableau = (int *)realloc(tableau, sizeof(int) * taille)) ==
→  NULL) {
    exit(EXIT_FAILURE);
```

```
}

for(i = 5; i < taille; ++i) {
    tableau[i] = 10 - i;
}
for(i = 0; i < taille; ++i) {
    if(i) { printf(", "); }
    printf("%d", tableau[i]);
}
printf("\n");
free(tableau);
tableau = NULL;
```

En sortie avec valgrind :

```
0, 1, 2, 3, 4
0, 1, 2, 3, 4, 5, 4, 3, 2, 1
==3610==
==3610== HEAP SUMMARY:
==3610==     in use at exit: 0 bytes in 0 blocks
==3610==   total heap usage: 3 allocs, 3 frees, 1,084 bytes
→  allocated
==3610==
==3610== All heap blocks were freed -- no leaks are possible
```

8.4 Résumé

Un pointeur est une type d'adresse : `type*` permet de sauvegarder l'adresse d'une variable `type`.

```
type variable;
&variable; /* adresse de variable */
type * adresse;
adresse = &variable; /* adresse prend comme valeur l'adresse de
→ variable */
*adresse; /* déréférence pour agir comme 'variable' */
```

Un tableau est l'adresse de son premier élément, manipuler un pointeur est équivalent à manipuler un tableau :

```
type tableau[];
type * pointeur = tableau;
pointeur[indice]; /* équivalent à tableau[indice] */
```

Un pointeur peut subir des opérations arithmétiques :

```
type * pointeur;
pointeur + 1; /* adresse de l'emplacement voisin suivant (au
→ sizeof(type) près) */

pointeur - 1; /* adresse de l'emplacement voisin précédent (au
→ sizeof(type) près) */

pointeur++; /* pointeur regarde l'élément suivant */
```

Réclamer de la mémoire :

```
type * pointeur = NULL;
pointeur = malloc(sizeof(type) * taille); /* alloue un tableau de
→ taille 'taille' */

pointeur = calloc(taille, sizeof(type)); /* alloue un tableau de
→ taille 'taille' et initialise les valeurs à 0 */

pointeur = realloc(pointeur, sizeof(type) * taille); /* change la
→ taille de la plage mémoire allouée */
```

Gestion d'allocation dynamique propre :

```
type * tableau = NULL;
if((tableau = (type *)malloc(sizeof(type) * taille)) == NULL) {
    /* traitement à appliquer, sauvegarde si possible */
    exit(EXIT_FAILURE);
}
/* utiliser tableau */
free(tableau);
tableau = NULL;
```

8.5 Entraînement

Exercice 70 (* Pointeurs sur des variables).

Vous êtes chargé d'effectuer un déménagement. Pour ceci, vous devrez modifier le code suivant mais seulement en ajoutant les caractères '*', '&' et des parenthèses :

01_demenagement.c : Code d'un déménagement

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int ma_maison = 42;
    int nouvelle_maison = 0;
    int mon_adresse = ma_maison;
    int nouvelle_adresse = nouvelle_maison;
    printf("[%c] Je sais où est ma maison ?\n",
           (mon_adresse == &ma_maison) ? 'V' : 'X');
    printf("[%c] Je sais où déménager ?\n",
           (nouvelle_adresse == &nouvelle_maison) ? 'V' : 'X');
    nouvelle_adresse = mon_adresse;
    mon_adresse = 0;
    printf("[%c] J'ai tout déménagé dans ma nouvelle maison
           ?\n",
           (ma_maison == 0) ? 'V' : 'X');
    printf("[%c] J'ai oublié des choses dans ma maison ?\n",
           (nouvelle_maison == 42) ? 'V' : 'X');
    mon_adresse = nouvelle_adresse;
    mon_adresse++;
    printf("[%c] J'habite à ma nouvelle adresse ?\n",
           (mon_adresse == &nouvelle_maison) ? 'V' : 'X');
    printf("[%c] J'ai ajouté des choses dans ma maison ?\n",
           (nouvelle_maison == 43) ? 'V' : 'X');
    exit(EXIT_SUCCESS);
}
```

Ceci devrait transformer toutes les X en V de sorte d'obtenir la sortie suivante :

```
[V] Je sais où est ma maison ?  
[V] Je sais où déménager ?  
[V] J'ai tout déménagé dans ma nouvelle maison ?  
[V] J'ai oublié des choses dans ma maison ?  
[V] J'habite à ma nouvelle adresse ?  
[V] J'ai ajouté des choses dans ma maison ?
```

Exercice 71 (★★ Pointinception).

Vérifiez que vous avez compris le système d'adressage et déréférencement en complétant le code suivant.

```
int valeur = 42;  
int * ptr = ...; /* TODO : pointe sur valeur */  
int ** pptr = ...; /* TODO : pointe sur ptr */  
int *** pppttr = ...; /* TODO : pointe sur pptr */  
printf("valeur via valeur : %d\n", valeur);  
printf("valeur via ptr : %d\n", ...);  
printf("valeur via pptr : %d\n", ...);  
printf("valeur via pppttr : %d\n", ...);
```

```
valeur via valeur : 42  
valeur via ptr : 42  
valeur via pptr : 42  
valeur via pppttr : 42
```

Exercice 72 (★★★ Construction liste extensible).

Bob aimerait écrire un programme qui gère une liste aussi grande qu'il le souhaite. Pour vous aider, Bob vous propose de commencer avec son code :

03_bob.c

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    const int ajouts = 10000;
    int * liste = malloc(sizeof(int));
    int * nouvelle_liste = NULL;
    int taille = 0;
    int i;
    int j;
    for(i = 0; i < ajouts; ++i) {
        liste[taille++] = i;
        nouvelle_liste = malloc(sizeof(int) * (taille + 1));
        for(j = 0; j < taille; ++j) {
            nouvelle_liste[j] = liste[j];
        }
        liste = nouvelle_liste;
    }
    printf("liste : [");
    for(i = 0; i < taille; ++i) {
        if(i) printf(", ");
        printf("%d", liste[i]);
    }
    printf("]\n");
    free(liste);
    exit(EXIT_SUCCESS);
}
```

1. Lorsque Bob lance `valgrind`, il semble qu'il y ait des fuites mémoires dans son programme :

```
==6552== HEAP SUMMARY:  
==6552==     in use at exit: 200,020,000 bytes in 10,000  
    ↳  blocks  
==6552==   total heap usage: 10,002 allocs, 2 frees,  
    ↳  200,061,028 bytes allocated
```

Pourriez-vous y remédier ?

2. Modifiez le code pour supprimer les variables `j` et `nouvelle_liste`.
3. Bob aimerait maintenant que sa liste puisse gérer 1 000 000 000 ajouts. Mais il semble que ré-allouer à chaque ajout semble ralentir le programme. Essayez d'allouer par blocs de 10, observez-vous une amélioration des performances ?

Exercice 73 (★★ Échanger dans un tableau).

Compléter le code suivant pour de sorte de :

- Définir les arguments de la fonction `echanger`.
- Écrire le code de la fonction `echanger`.
- Compléter l'appel à la fonction `echanger`.
- Afficher une valeur à un indice dans sa version pointeur.

```
#include <stdio.h>
#include <stdlib.h>

void echanger(/* TODO : arguments à échanger */) {
    /* TODO : code pour échanger valeurs */
}

int main() {
    /* interdiction d'accéder à un élément par tableau[indice] */
    /* interdiction d'accéder à une adresse par &tableau[indice] */
    /* TODO : utiliser la version pointeur : */
    int tableau[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int indice_first = 2;
    int indice_second = 4;
    echanger(/* tableau à indice_first */, /* tableau à
        → indice_second */);
    int i;
    printf("tableau : [");
    for(i = 0; i < 10; ++i) {
        if(i) printf(", ");
        printf("%d", /* tableau à i */);
    }
    printf("]\n");
    exit(EXIT_SUCCESS);
}
```

Exercice 74 (★★★ Tableau de variables).

Compléter le code suivant pour de sorte de :

- Définir l'argument de la fonction `print_values`.
- Afficher l'adresse de chaque adresse (élément du tableau) et son contenu.
- Définir un tableau d'adresses d'entiers.
- Incrémenter une valeur depuis son adresse.
- Expliquer un comportement.

```
#include <stdio.h>
#include <stdlib.h>

void print_values(/* TODO : type */ variables) {
    printf("values : [\n");
    for(; *variables; ++variables) {
        printf("\t0x%p : %d\n",
            /* TODO : afficher les éléments de variables : */
            /* adresse et valeur */);
    }
    printf("]\n");
}

int main() {
    int a = 1;
    int b = 2;
    int c = 3;
    int i = 0;
    /* TODO : type */ variables[] = {
        &i, &a, &b, &c, NULL
    };
    print_values(variables);
    a = 42;
    for(i = 0; variables[i] != NULL; ++i) {
        /* TODO : ajouter 1 à l'adresse pointée par variables[i] */
    }
    print_values(variables);
    /* TODO : Pourquoi a vaut 42 ? */
    exit(EXIT_SUCCESS);
}
```

Exercice 75 (★★ Adresse sur un entier).

Compléter le code suivant pour de sorte de :

- Sauvegarder une adresse sous un entier.
- Déréférencer une adresse exprimée par un entier.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    /* TODO : faire fonctionner le code sans changer les types de
       → variable et pointeur */
    int variable = 42;
    unsigned long pointeur = &variable;
    *pointeur = 1337;
    /* TODO : variable doit valoir 1337 */
    printf("%d = 1337 ?\n", variable);
    exit(EXIT_SUCCESS);
}
```

Exercice 76 (★★ Dupliquer chaîne de caractère via arithmétique de pointeurs).

Compléter le code suivant pour de sorte de :

- Déterminer la fin de la chaîne.
- Dupliquer chaque caractère en fin de la chaîne fournie en argument.

```
#include <stdio.h>
#include <stdlib.h>
/* interdiction d'importer string.h */

void chaine_double(char * chaine) {
    char * aux;
    /* TODO : aux peut être utilisée */
    char * end;
    /* TODO : end doit permettre la copie des caractères de chaine
       → */
}
```

```

int main() {
    char chaine[40] = "Hello ! ";
    printf("x 1 : %s\n", chaine);
    chaine_double(chaine);
    printf("x 2 : %s\n", chaine);
    chaine_double(chaine);
    printf("x 4 : %s\n", chaine);
    exit(EXIT_SUCCESS);
}

```

Exercice 77 (★★ Visualiser les sauts d'octets et interprétation des valeurs).

Réaliser les appels aux fonctions données tels que demandés.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void read_char(unsigned char * ptr, int taille) {
    int i;
    printf("[");
    for(i = 0; i < taille; ++i, ++ptr) {
        if(i) printf(", ");
        printf("0x%x", *ptr);
    }
    printf("]\n");
}

void read_int(unsigned int * ptr, int taille) {
    int i;
    printf("[");
    for(i = 0; i < taille; ++i, ++ptr) {
        if(i) printf(", ");
        printf("0x%x", *ptr);
    }
    printf("]\n");
}

int main() {
    char chaine[] = "Exemple de texte";
    int values[] = {0xFEEDCAFE, 0xCODEF00D};

```

```
/* TODO : appeler read_char sur chaine */
/* TODO : appeler read_int sur values */
/* TODO : appeler read_int sur chaine */
/* TODO : appeler read_char sur values */
exit(EXIT_SUCCESS);
}
```

Exercice 78 (★★ Concept de pointeur).

Pour illustrer le concept de pointeurs, on vous propose de comprendre le code suivant proposé et d'implémenter les fonctions correspondantes sans modifier le code de la fonction `main` :

```
#include <stdio.h>
#include <stdlib.h>

#define NB_MAISONS 7

void afficher_maison(int * maison);
void afficher_maisons(int * maisons, int nombre);
int * adresse_maison(int * maisons, int numero);
int vider_maison(int * maison);
void vider_camion(int * camion, int * maison);

int main() {
    int * demenageur = NULL;
    int camion = 0;
    int maisons[NB_MAISONS];
    int i;

    for(i = 0; i < NB_MAISONS; ++i) {
        maisons[i] = i + 1;
    }
    afficher_maisons(maisons, NB_MAISONS);

    printf("camion -> %d\n", camion);

    demenageur = adresse_maison(maisons, 3);
    afficher_maison(demenageur);
    camion = vider_maison(demenageur);
    afficher_maison(demenageur);

    printf("camion -> %d\n", camion);

    demenageur = adresse_maison(maisons, 5);
    afficher_maison(demenageur);
    vider_camion(&camion, demenageur);
    afficher_maison(demenageur);
```

```
printf("camion -> %d\n", camion);
afficher_maisons(maisons, NB_MAISONS);

exit(EXIT_SUCCESS);
}
```

Le résultat attendu devrait être similaire à celui de la sortie suivante :

```
[0] : 0x7ffddb7ea830 -> 1
[1] : 0x7ffddb7ea834 -> 2
[2] : 0x7ffddb7ea838 -> 3
[3] : 0x7ffddb7ea83c -> 4
[4] : 0x7ffddb7ea840 -> 5
[5] : 0x7ffddb7ea844 -> 6
[6] : 0x7ffddb7ea848 -> 7
camion -> 0
0x7ffddb7ea83c -> 4
0x7ffddb7ea83c -> 0
camion -> 4
0x7ffddb7ea844 -> 6
0x7ffddb7ea844 -> 10
camion -> 0
[0] : 0x7ffddb7ea830 -> 1
[1] : 0x7ffddb7ea834 -> 2
[2] : 0x7ffddb7ea838 -> 3
[3] : 0x7ffddb7ea83c -> 0
[4] : 0x7ffddb7ea840 -> 5
[5] : 0x7ffddb7ea844 -> 10
[6] : 0x7ffddb7ea848 -> 7
```

Exercice 79 (★★★ Concaténation de chaînes de caractères).

Alice une habituée du langage python essaie de concaténer des chaînes de caractères avec l'opérateur '+' mais le compilateur refuse de faire ce qu'elle veut. Proposez à Alice une manière de faire ce qu'elle souhaite en langage C à partir de son code :

02_alice.c

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    char nom[100], prenom[50];
    char * full_name = NULL;
    printf("Bonjour, entrez votre nom et prénom : ");
    scanf("%s %s", nom, prenom);
    full_name = prenom + ' ' + nom;
    printf("Vous êtes donc %s !\n", full_name);
    exit(EXIT_SUCCESS);
}
```

Exercice 80 (★★★ Gestion d'une liste de noms).

Charlie souhaite gérer une liste de noms et a besoin de votre aide. C'est-à-dire qu'il aimerait pouvoir y ajouter des mots, y rechercher les mots et les supprimer. Ceci pourrait donner la sortie suivante :

```
>>> help
[Info] : "help" action inconnue.
Actions possibles :
- ajouter [mot]
- supprimer [mot]
- rechercher [mot]
- afficher
- quitter
>>> ajouter Pikachu
[Info] : ajout réussi.
>>> afficher
Mots : {
    - "Pikachu"
}
>>> ajouter Pichu
[Info] : ajout réussi.
>>> rechercher Pikachu
[Info] : Pikachu trouvé.
>>> rechercher Tortank
[Info] : Tortank non trouvé.
>>> afficher
Mots : {
    - "Pikachu"
    - "Pichu"
}
>>> supprimer Pikachu
[Info] : suppression réussie.
>>> afficher
Mots : {
    - "Pichu"
}
>>> quitter
```

Exercice 81 (**★★★** Tableau dynamique à deux dimensions).

Bob a vu Charlie proposer un code qui permet la construction allouée d'une grille et son affichage. A priori, ceci peut ressembler à aux sorties suivantes :

```
Entrez largeur et hauteur de la grille : 10 5
#-----#
-#-----#-
--#####--#
-#-----#-
#-----#
```

```
Entrez largeur et hauteur de la grille : 5 10
#---#
-#-#-
--#--#
--#--#
--#--#
--#--#
--#--#
--#--#
-#-#-
#---#
```

```
Entrez largeur et hauteur de la grille : 5 5
#---#
-#-#-
--#--#
-#-#-
#---#
```

A priori, ceci pourrait se coder avec un tableau à deux dimensions alloué dynamiquement. Bob imagine qu'il peut avoir un tableau de pointeurs et pour chaque pointeur un tableau alloué dynamiquement. Charlie lui indique que ça pourrait se faire en deux allocations peu importe les dimensions de la grille. Bob et Charlie vous défient de proposer une solution à leur problème en codant les fonctions déclarées dans le code suivant :

```
#include <stdio.h>
#include <stdlib.h>

int ** allouer_grille(int largeur, int hauteur);
void afficher(int ** grille, int largeur, int hauteur);
void fill(int ** grille, int largeur, int hauteur);
void free_grille(int *** grille);

int main() {
    int largeur, hauteur;
    int ** grille = NULL;

    printf("Entrez largeur et hauteur de la grille : ");
    scanf("%d %d", &largeur, &hauteur);

    grille = allouer_grille(largeur, hauteur);
    fill(grille, largeur, hauteur);
    afficher(grille, largeur, hauteur);
    free_grille(&grille);
    exit(EXIT_SUCCESS);
}
```

Exercice 82 (∞ Labyrinthe).

Bob aimeraient coder un labyrinthe, il a essayé de tester une méthode faite maison pour creuser le labyrinthe et le générer automatiquement. Mais plus rien ne fonctionne. Robert a rapidement regardé et lui a indiqué quelques étapes à suivre pour réussir. Bob fait donc appel à vous pour finaliser son projet :

- Allouer une grille (tableau à deux dimensions) dans la fonction `grille_creer`.
- Coder la fonction `in_grille` (vérifie que les coordonnées ne sortent pas de la zone définie pour la grille).
- Coder les fonctions `ecran_depuis_camera` et `camera_depuis_ecran` pour rendre la caméra utilisable.

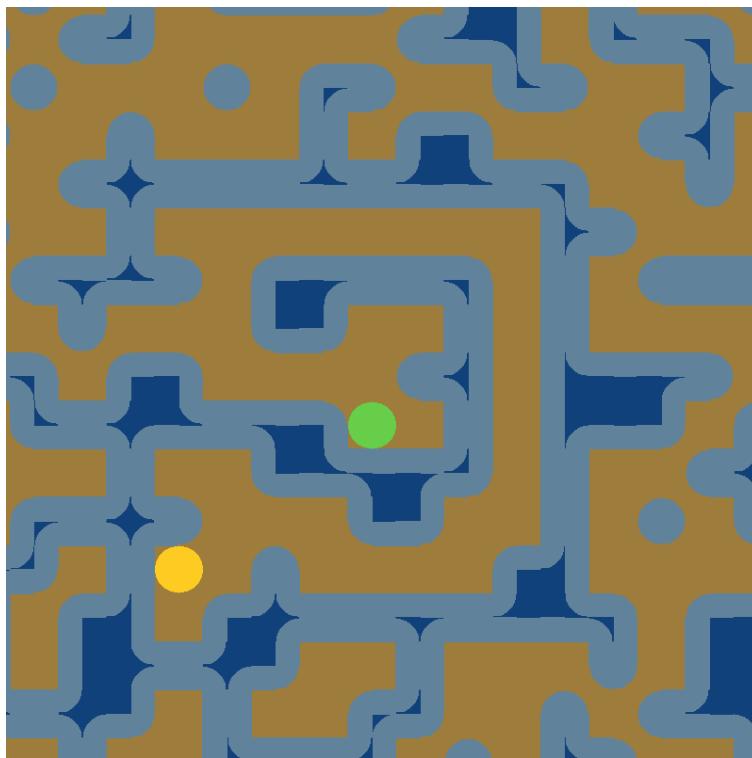
Une fois cette première version fonctionnelle, Robert vous indique que la méthode de Bob telle que codée n'est pas viable. En effet, pour une taille de grille de 1 000 sur 1 000, sa méthode exploserait la pile en appels récursifs. En effet, `valgrind` afficherait des messages d'erreur comme :

```
==10690== Stack overflow in thread #1: can't grow stack to 0x1ffe801000
==10690== Stack overflow in thread #1: can't grow stack to 0x1ffe801000
==10690== Can't extend stack to 0x1ffe8010a8 during signal delivery for thread 1:
==10690==   no stack segment
==10690==
==10690== Process terminating with default action of signal 11 (SIGSEGV)
==10690==  Access not within mapped region at address 0x1FFE8010A8
==10690== Stack overflow in thread #1: can't grow stack to 0x1ffe801000
==10690==   at 0x109275: creuser (04_laby.c:71)
```

Robert vous informe que dans ce type de cas, il vaut mieux dérécursiver les appels de la fonction à l'aide d'une pile que vous géreriez vous-même. Une fois que vous aurez réussi à générer une grille 10 000 sur 10 000, vous pourriez finaliser votre projet :

- Optimiser l'affichage pour qu'il fonctionne en vue proche et vue lointaine.
- Déplacer le joueur au clavier.
- Proposer un affichage stylisé des murs.

Une proposition d'un affichage stylisé pourrait par exemple ressembler à l'image suivante :



Bob vous a laissé le code suivant pour démarrer :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <assert.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 800;
const char * titre = "ESGI labyrinthe";

/* Mon jeu va être génial, mais il y a encore quelques bugs. */
```

```

/* Merci de les corriger. Bob. */

/* TODO : allouer une grille (fonction grille_creer) */
/* TODO : coder in_grille */
/* Normalement, ça affiche un labyrinthe. */
/* TODO : corriger la caméra. */
/* TODO : tester pour :
/* - grille_largeur = 1000 */
/* - grille_hauteur = 1000 */
/* L'erreur vient d'appels récursifs qui explosent la pile. */
/* TODO : optimiser les appels récursifs de creuser . */
/* TODO : adapter le code pour des grilles 10000 x 10000 . */
/* TODO : optimiser l'affichage (vue proche et vue loin) . */
/* TODO : déplacer le joueur au clavier dans le labyrinthe . */
/* TODO : s'amuser à aller plus loin si vous le voulez ! */

/* Position de la caméra et échelle de la vue : */
float cx, cy, cz;

int player_x, player_y;
int exit_x, exit_y;

SDL_Surface * ecran = NULL;

char ** grille = NULL;
int grille_largeur = 100;
int grille_hauteur = 100;

/* TODO permettre déplacement caméra à la souris : */
double ecran_depuis_camera_x(double x) {
    return x / cz + cx;
}

double ecran_depuis_camera_y(double y) {
    return y / cz + cy;
}

double ecran_depuis_camera_z(double z) {
    return z / cz;
}

double camera_depuis_ecran_x(double x) {
    return x * cz - cx;
}

```

```
double camera_depuis_ecran_y(double y) {
    return y * cz - cy;
}

double camera_depuis_ecran_z(double z) {
    return z * cz;
}

void grille_creer(int width, int height) {
    /* TODO : allouer le tableau à deux dimensions : */
    grille = calloc(width, height);
    grille_largeur = width;
    grille_hauteur = height;
}

int in_grille(int x, int y) {
    /* TODO : vérifier si deux coordonnées sont dans la grille : */
    return 0;
}

void creuser(int x, int y) {
    /* TODO : corriger les erreurs de segmentation : */
    if(! in_grille(x, y)) return ;
    if(grille[x][y] != '#') return ;
    int link4 = 0;
    int link8 = 0;
    int i, j;
    int dx = 0, dy = 0;
    int tmp;
    for(i = -1; i <= 1; ++i) {
        for(j = -1; j <= 1; ++j) {
            if(! in_grille(x + i, y + j)) continue;
            if(i == j) continue;
            if(grille[x + i][y + j] == '.') ++link8;
            if(abs(i) + abs(j) != 1) continue;
            if(grille[x + i][y + j] == '.') ++link4;
        }
    }

    if(link4 > 1) return ;
    if(link8 > 2) return ;

    grille[x][y] = '.';
}

/* TODO : améliorer les performances : */
```

```

int perm[4] = {0, 1, 2, 3};
for(i = 0; i < 20; ++i) {
    int a = rand() % 4, b = rand() % 4;
    tmp = perm[a];
    perm[a] = perm[b];
    perm[b] = tmp;
}
for(i = 0; i < 4; ++i) {
    switch(perm[i]) {
        case 0 : dx = 1; dy = 0; break;
        case 1 : dx = 0; dy = 1; break;
        case 2 : dx = -1; dy = 0; break;
        case 3 : dx = 0; dy = -1; break;
    }
    creuser(x + dx, y + dy);
}
}

void grille_generer(int width, int height) {
grille_creer(width, height);
/* TODO : mettre tous les éléments de la grille à '#'.*/
/* TODO : améliorer les tailles de grilles : */
creuser(player_x, player_y);
exit_x = rand() % width;
exit_y = rand() % height;
}

void init() {
player_x = grille_largeur / 2;
player_y = grille_hauteur / 2;
clock_t start, end;
start = clock();
grille_generer(grille_largeur, grille_hauteur);
end = clock();
printf("Génération d'une grille %dx%d en %g s\n", grille_largeur,
   → grille_hauteur, (double)(end - start) / CLOCKS_PER_SEC);
}

int check_finish() {
/* TODO : indiquer si le joueur a atteint la sortie. */
/* TODO : générer le niveau suivant si terminé. */
return 0;
}

int get_grille(int x, int y) {

```

```
/* TODO : renvoyer l'élément présent à la coordonnée (mur ou chemin) */
return (rand() % 2) ? '#' : '.';
}

int can_move(int x, int y) {
    /* TODO : valider ou non un déplacement vers des coordonnées. */
    return 1;
}

void afficher_grille() {
    /* TODO : optimiser l'affichage. */
    int x, y;
    for(x = 0; x < grille_largeur; ++x) {
        for(y = 0; y < grille_hauteur; ++y) {
            switch(get_grille(x, y)) {
                case '#': boxRGBA(ecran,
                    ecran_depuis_camera_x(x - 0.5), ecran_depuis_camera_y(y - 0.5),
                    ecran_depuis_camera_x(x + 0.5), ecran_depuis_camera_y(y + 0.5),
                    51, 51, 51, 255); break;
                case '.': boxRGBA(ecran,
                    ecran_depuis_camera_x(x - 0.5), ecran_depuis_camera_y(y - 0.5),
                    ecran_depuis_camera_x(x + 0.5), ecran_depuis_camera_y(y + 0.5),
                    204, 204, 204, 255); break;
                default : boxRGBA(ecran,
                    ecran_depuis_camera_x(x - 0.5), ecran_depuis_camera_y(y - 0.5),
                    ecran_depuis_camera_x(x + 0.5), ecran_depuis_camera_y(y + 0.5),
                    25, 25, 25, 255); break;
            }
        }
    }
}

void affichage() {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 51));
    afficher_grille();
    filledCircleRGBA(ecran,
        ecran_depuis_camera_x(player_x), ecran_depuis_camera_y(player_y),
        ecran_depuis_camera_z(0.5),
        25, 153, 25, 255);
    filledCircleRGBA(ecran,
        ecran_depuis_camera_x(exit_x), ecran_depuis_camera_y(exit_y),
        ecran_depuis_camera_z(0.5),
        25, 153, 153, 255);
}
```

```

int main(int argc, char * argv[]) {
    if(argc >= 3) {
        grille_largeur = atoi(argv[1]);
        grille_hauteur = atoi(argv[2]);
    }
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);
    init();

    int active = 1;
    SDL_Event event;
    int refresh = 1;
    int right_grab = 0;

    /* Placement du joueur au centre de la fenêtre : */

    cx = grille_largeur / 2;
    cy = grille_hauteur / 2;
    cz = 5;

    SDL_EnableKeyRepeat(50, 50);

    while(active) {

        if(refresh) {
            affichage();
            SDL_Flip(ecran);
            refresh = 0;
        }

        while(SDL_PollEvent(&event)) {

            switch(event.type) {

```

```
/* Utilisateur clique sur la croix de la fenêtre : */
case SDL_QUIT : {
    active = 0;
} break;

/* Utilisateur enfonce une touche du clavier : */
case SDL_KEYDOWN : {
    switch(event.key.keysym.sym) {
        /* Touche Echap : */
        case SDLK_ESCAPE : {
            active = 0;
        } break;

        case SDLK_SPACE : {
            cx = player_x;
            cy = player_y;
            refresh = 1;
        } break;

        case SDLK_z :
        case SDLK_UP : {
            if(can_move(player_x, player_y - 1)) {
                --player_y;
                refresh = 1;
                if(ecran_depuis_camera_y(player_y) < 0) {
                    cy -= camera_depuis_ecran_z(hauteur);
                }
            }
        } break;

        case SDLK_s :
        case SDLK_DOWN : {
            if(can_move(player_x, player_y + 1)) {
                ++player_y;
                refresh = 1;
                if(ecran_depuis_camera_y(player_y) > hauteur) {
                    cy += camera_depuis_ecran_z(hauteur);
                }
            }
        } break;

        case SDLK_q :
        case SDLK_LEFT : {
            if(can_move(player_x - 1, player_y)) {
                --player_x;
            }
        } break;
    }
}
```

```

refresh = 1;
if(ecran_depuis_camera_x(player_x) < 0) {
    cx -= camera_depuis_ecran_z(largeur);
}
}

} break;

case SDLK_d :
case SDLK_RIGHT : {
    if(can_move(player_x + 1, player_y)) {
        ++player_x;
        refresh = 1;
        if(ecran_depuis_camera_x(player_x) > largeur) {
            cx += camera_depuis_ecran_z(largeur);
        }
    }
} break;
}

} break;

case SDL_MOUSEBUTTONDOWN : {
switch(event.button.button) {
    case SDL_BUTTON_WHEELUP : {
        cz *= 0.8;
        if(cz < 1e-9) {
            cz = 1e-9;
        }
        refresh = 1;
    } break;
}

case SDL_BUTTON_WHEELDOWN : {
    cz /= 0.8;
    if(cz > 1e9) {
        cz = 1e9;
    }
    refresh = 1;
} break;

case SDL_BUTTON_RIGHT : {
    right_grab = 1;
} break;
}

} break;

case SDL_MOUSEBUTTONUP : {

```

```
switch(event.button.button) {
    case SDL_BUTTON_RIGHT : {
        right_grab = 0;
    } break;
}

case SDL_MOUSEMOTION : {
    if(right_grab) {
        cx += camera_depuis_ecran_z(-event.motion.xrel);
        cy += camera_depuis_ecran_z(-event.motion.yrel);
        refresh = 1;
    }
} break;
}

if(check_finish()) {
    cx = player_x;
    cy = player_y;
    refresh = 1;
}

SDL_Delay(1000 / 60);
}

/* TODO : libérer grille. */

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```


Quatrième partie

Notions avancées

Table des matières

9 Consolidation des bases	303
9.1 Remise dans le bain rapide	303
9.1.1 Un programme en C	303
9.1.2 Variables	304
9.1.3 Opérations	305
9.1.4 Tests et disjonctions	306
9.1.5 Boucles	307
9.1.6 Fonctions	308
9.1.7 Tableaux	309
9.1.8 Pointeurs	310
9.2 Entraînement	313
10 Fichiers	325
10.1 Ouverture et création de fichiers	325
10.2 Lecture et écriture	326
10.2.1 Avec fonctions formatées	326
10.2.2 Par caractère	328
10.2.3 Par bloc mémoire	329
10.3 Se déplacer dans un fichier	331
10.4 Flux standards d'entrées et sorties	333
10.4.1 stdout	333
10.4.2 stdin	333
10.4.3 stderr	334
10.5 Résumé	335
10.6 Entraînement	337

11 Structures	343
11.1 Typedef	343
11.2 Structures	344
11.3 Définition	345
11.4 Avec des pointeurs	346
11.4.1 Champs de bits	349
11.5 Unions	351
11.6 Énumérations	352
11.7 Résumé	355
11.8 Entraînement	356
12 Programmation modulaire	383
12.1 Retour sur la compilation	383
12.1.1 Externaliser une fonction	383
12.1.2 Compilation séparée	385
12.2 Directives préprocesseur	387
12.2.1 include	388
12.2.2 define	389
12.2.3 conditionnement	394
12.2.4 Module	395
12.3 Makefiles	398
12.3.1 Concept	398
12.3.2 Possibilités	398
12.4 Résumé	403
12.5 Entraînement	405
13 Types génériques	419
13.1 Pointeurs de fonctions	420
13.1.1 Définition	420
13.1.2 Appel	421
13.1.3 Constructions plus complexes	422
13.1.4 Typedef	424
13.2 Pointeurs génériques	426
13.2.1 Intérêt	427
13.2.2 Exemple avec qsort	427
13.2.3 Manipulation	429
13.3 Fonctions variadiques	432

13.4 Bibliothèques dynamiques	434
13.4.1 Compiler sa bibliothèque	435
13.4.2 Utiliser sa bibliothèque à la compilation	436
13.4.3 Charger sa bibliothèque à l'exécution	437
13.5 Résumé	440
13.6 Entraînement	442
14 Bonus : Opérations bit-à-bit	473
14.1 Décalages	473
14.1.1 À gauche	473
14.1.2 À droite	474
14.2 Négation	474
14.3 Opérateurs booléens	475
14.3.1 et	475
14.3.2 ou inclusif	476
14.3.3 ou exclusif : xor	477
14.4 Résumé	479
14.5 Entraînement	481

9 Consolidation des bases

Les vacances sont passées, mais nous revoilà à étudier le langage C! On peut faire quelques rappels rapides avant d'avancer.

9.1 Remise dans le bain rapide

9.1.1 Un programme en C

Dans un programme en C :

1. Vous commencez par importer les bibliothèques dont les fonctionnalités vous seront utiles avec `#include`.
2. Vous listez les déclarations des fonctions que vous pourriez vouloir définir.
3. Vous écrivez les instructions à suivre au lancement de votre application dans la fonction `main`.
4. Vous listez les déclarations de vos fonctions.

Rappelez vous, ça ressemble à ça :

```
#include <stdio.h>
#include <stdlib.h>
/* ... */

/* Déclaration des fonctions */

int main() {

    /* Instructions de l'application */

    exit(EXIT_SUCCESS);
}

/* Définition des fonctions */
```

Pour compiler un tel programme depuis un fichier source `main.c`. On peut utiliser `gcc` et préciser le nom du programme avec l'option `-o` :

```
gcc -o monProgramme main.c
./monProgramme
```

9.1.2 Variables

En langage C, les variables sont typées. Les types atomiques sont :

Entiers :

```
char /* 1 octet */
short /* 2 octets */
int /* 4 octets */
long /* 8 octets */
unsigned type /* positif */
```

Flottants :

```
float /* 4 octets */
double /* 8 octets */
long double /* 16 octets */
```

Vous devez les utiliser lorsque vous déclarez une variable ou pour forcer un changement de type :

```
unsigned int valeurPositive = 42; /* Affectation de 42 */
valeurPositive = 0x2a; /* Réaffectation en hexadécimal */
```

Pour imprimer des caractères dans le terminal et afficher les valeurs de vos variables vous avez la fonction d'affichage formaté `printf` :

```
printf("caractere : \'%c\'\n", '0');
printf("entier : %d\n", 42);
printf("hexadécimal : %x\n", 0x2a);
printf("entier positif : %u\n", 3000000000u);
printf("entier long : %ld\n", 42000000000);
printf("flottant : %g\n", 3.14);
printf("adresse : %p\n", NULL);
```

Pour lire les caractères saisis par l'utilisateur dans son terminal, vous avez `scanf`. Notez que vous devrez passer une adresse à `scanf` pour réussir votre affectation :

```
int entier;
scanf("%d", &entier);
float flottant;
scanf("%f", &flottant);
char caractere;
scanf(" %c", &caractere);
```

9.1.3 Opérations

Souvent il va être intéressant de combiner les valeurs en utilisant des opérateurs. Les opérateurs classiques sont les suivants :

```
/*first et second deux variables de type entier ou flottant*/
first + second /* addition */
first - second /* soustraction */
first * second /* multiplication */
first / second /* division */
first % second /* modulo : entiers */
```

Notez que dans le cas d'entiers, vous seriez amenés à utiliser une coercition pour gérer un dépassement de capacité d'un `int` ou pour réaliser une division avec résultat flottant :

```
int first, second;
/* ... */
long multiplication = (long)first * second;
float division = (float)first / second;
```

En langage C, il existe des raccourcis pour incrémenter une valeur ou utiliser un opérateur :

```
int valeur = 1;
valeur = valeur + 1; /* ajoute 1 */
valeur += 1;           /* ajoute 1 */
valeur++;             /* ajoute 1 */
++valeur;              /* ajoute 1 */
```

9.1.4 Tests et disjonctions

Il est possible de réaliser une disjonction de cas sous condition à l'aide d'un **if-else** :

```
int first, second;
scanf("%d %d", &first, &second);
if(first > second) {
    printf("%d est plus grand.\n", first);
} else if(first < second) {
    printf("%d est plus grand.\n", second);
} else {
    printf("les deux sont égaux.\n");
}
```

On différencie le test d'égalité == de l'affectation =. Les opérateurs de comparaison sont les suivants :

<i>/* égalité : */</i>	<i>/* différence : */</i>
a == b	a != b
<i>/* plus petit strict : */</i>	<i>/* plus grand strict : */</i>
a < b	a > b
<i>/* plus petit ou égal : */</i>	<i>/* plus grand ou égal : */</i>
a <= b	a >= b

Pour assembler les valeurs de vérité renvoyées par ces opérateurs, on utilise les opérateurs suivants :

```
a && b /* vrai lorsque les deux le sont */
a || b /* vrai lorsque d'un l'est */
! a /* vrai lorsque faux et réciproquement */
```

Il est possible de synthétiser une affectation dans un **if-else** à l'aide d'une opération ternaire :

```
if(a < b) {
    res = a;
} else {
    res = b;
}
```

\Leftrightarrow

```
res = (a < b) ? a : b;
```

Pour des conditions simples d'égalité à une constante, il est possible d'utiliser un **switch** pour se brancher à un bloc correspondant :

```
switch(expression) {
    case 1 : {
        /* instructions */
    } break;
    case 2 : {
        /* instructions */
    } break;
    default : {
        /* instructions */
    }
}
```

9.1.5 Boucles

Il est possible d'automatiser la répétition d'instructions à l'aide de boucles.

On utilise :

- **while** pour répéter les instructions tant qu'une condition est vraie.

```
while(condition) {
    /* instructions */
}
```

- **do-while** pour répéter à nouveau les instructions lorsqu'une condition est vérifiée.

```
do {
    /* instructions */
} while(condition);
```

- **for** lorsque la boucle while peut s'écrire avec une initialisation et une évolution des données utilisées pour la condition.

```
for(initialisation; condition; evolution) {
    /* instructions */
}
```

Il est aussi possible de relancer la boucle prématurément à l'aide du mot-clé **continue** ou de la stopper avant vérification de la condition par le mot-clé **break**.

9.1.6 Fonctions

Une fonction se définit par un type de retour, un nom d'appel et des paramètres :

```
typeRetour nomFonction(/* paramètres */) {
    /* instructions */
}

/* Exemple de fonction d'addition d'entiers : */
int addition(int first, int second) {
    int res; /* variable locale à la fonction */
    res = first + second;
    return res; /* renvoi lors de l'appel */
}
```

Cette fonction peut être déclarée en amont par sa signature (type de retour, nom et type des paramètres) et appelée par son nom :

```
/* Exemple de fonction d'addition d'entiers : */
/* déclaration */
int addition(int, int);
```

```

int main() {
    /* appel */
    int deux = addition(1, 1);
    exit(EXIT_SUCCESS);
}

/* définition */
int addition(int first, int second) {
    return first + second;
}

```

9.1.7 Tableaux

Un tableau est l'outil en langage C qui permet de créer une liste de taille donnée pour une type de variable que l'on souhaite répéter :

```

type tableau[TAILLE_CONSTANTE];
type tableau[] = {valeur1, valeur2, ..., valeurN};

int liste[] = {1, 2, 3};
liste[1]; /* accès au second élément : d'indice 1 */

```

Les tableaux sont utilisés pour gérer les chaînes de caractères (se terminant par un marque de fin '\0') :

```

char chaine[] = "Hello ESGI !";
printf("%s\n", chaine); /* affichage de chaine */
scanf("%s", chaine); /* lecture d'un mot au clavier */

```

Il est possible de gérer des tableaux à plusieurs dimensions :

```

/* tableau à deux dimensions */
int grille[HAUTEUR][LARGEUR] = {
    {0, 0, 0, 0},
    {0, 1, 2, 0},
    {0, 0, 0, 0}
}

```

```

};

grille[ligne][colonne]; /* accès au tableau */

/* passage d'un tableau à deux dimensions */
void afficherGrille(int largeur, int hauteur,
    grille[hauteur][largeur]) {
    /* instructions */
}

```

9.1.8 Pointeurs

Toute donnée en mémoire possède une adresse, les pointeurs sont l'outil qui permet d'utiliser ces adresses dans votre programme. Nous avons vu que nous pouvons récupérer l'adresse d'une variable à l'aide de l'opération unaire `&`. Pour sauvegarder cette adresse dans une variable, on construira une pointeur sur cette variable dont le type prendra une étoile par rapport à celui de la variable. L'accès à la donnée pointée se fera ensuite à l'aide d'un déréférencement par l'opérateur unaire `*` :

```

int variable = 42;
int * pointeur = &variable;
*pointeur = 1337;
printf("%d\n", variable); /* affiche 1337 */

```

Pour rappel, un tableau correspond à l'adresse de son premier élément (d'indice 0), les autres étant alignés en mémoire après celui-ci. La manipulation d'un tableau peut aussi s'appliquer avec un pointeur :

```

int tableau[] = {1, 2, 3, -1};
int * pointeur = tableau;
int i;
for(i = 0; pointeur[i] >= 0; ++i) {
    printf("%d\n", pointeur[i]);
}

```

À noter qu'un pointeur peut changer de valeur (pointer vers une autre adresse), ce qui permet par exemple de jouer avec l'arithmétique des pointeurs :

```

char texte[] = "Hello !";
char * pointeur = NULL;
for(pointeur = texte; *pointeur != '\0'; ++pointeur) {
    if(*pointeur >= 'a' && *pointeur <= 'z')
        *pointeur += 'A' - 'a';
}
printf("%s\n", texte); /* affiche "HELLO !" */

```

L'un des grands intérêts des pointeurs est de pouvoir manipuler des plages mémoire allouées dynamiquement : de taille décidée à l'exécution du programme.

```

float * notes = NULL;
float somme = 0;
int nombre;
int i;
printf("Combien de CC ? ");
scanf("%d", &nombre);
if(nombre <= 0) {
    printf("Pas de notes pas de moyenne.\n");
    exit(EXIT_FAILURE);
}
/* allocation dynamique depuis le nombre donné par l'utilisateur
   */
if((notes = (float *)malloc(sizeof(float) * nombre)) == NULL) {
    printf("Erreur d'allocation.\n");
    exit(EXIT_FAILURE);
}
for(i = 0; i < nombre; ++i) {
    scanf("%f", notes + i);
    somme += notes[i];
}
printf("La moyenne de ");
for(i = 0; i < nombre; ++i) {
    if(i && i == nombre - 1) printf(" et ");
    else if(i > 0) printf(", ");
}

```

```
    printf("%g", notes[i]);
}
printf(" est %g\n", somme / nombre);

free(notes);
notes = NULL;
exit(EXIT_SUCCESS);
```

Les principales fonctions d'allocation sont les suivantes :

```
/* allouer une plage mémoire */
malloc(/*taille mémoire en octets*/)

/* allouer un tableau avec chaque élément à 0 */
calloc(/*taille tableau*/, /*taille élément*/)

/* modifier la taille d'une plage allouée */
realloc(/*plage à modifier*/, /*nouvelle taille en octets*/)
```

À jour et prêt pour la suite ? Pas de panique, on fait des exercices là dessus pour se remettre dans le bain.

9.2 Entraînement

Exercice 83 (** Lecture d'arguments en ligne de commande).

Compléter le code suivant pour de sorte de :

- Définir les arguments de la fonction `main` pour récupérer la ligne de commande lancée.
- Afficher les arguments lus.

```
#include <stdio.h>
#include <stdlib.h>

int main(/* TODO : arguments */) {
    /* TODO : lister les arguments du programme */
    exit(EXIT_SUCCESS);
}
```

Exercice 84 (*** Options depuis ligne de commande).

Compléter le code suivant pour de sorte de :

- Lire les arguments saisis lors de la ligne de commande.
- Déterminer un système d'options depuis ces arguments pour affecter les variables données.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(/* TODO : arguments du programme */) {
    int first = 0;
    int second = 0;
    int x = 0, y = 0;
    /* TODO : changer les valeurs de first, second et (x, y) depuis
       des options */
    printf("first = %d\nsecond = %d\ncoords = (%d, %d)\n", first,
           second, x, y);
    exit(EXIT_SUCCESS);
}
```

Exercice 85 (★ Allouer dynamiquement une variable).

Compléter le code suivant pour de sorte de :

- Allouer un emplacement mémoire pour un entier.
- Assigner une valeur à cet emplacement mémoire.
- Lire le contenu à cet emplacement mémoire.
- Libérer l'emplacement mémoire.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int * variable;
    /* TODO : allouer variable et mettre son contenu à 42 */
    printf("%d\n", /* TODO : contenu de variable */);
    /* TODO : pensez à libérer variable */
    exit(EXIT_SUCCESS);
}
```

Exercice 86 (★★ Allouer dynamiquement tableau).

Compléter le code suivant pour de sorte de :

- Allouer un tableau dynamique de réels.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    double * tableau = NULL;
    int taille;
    printf("Combien d'éléments ? ");
    scanf("%d", &taille);
    /* allouer 'tableau' initialisé par des valeurs à 0 de 'taille'
       → donnée */
    int i;
    printf("Entrez leurs valeurs : ");
    for(i = 0; i < taille; ++i) {
        scanf("%lf", tableau + i);
```

```

}
printf("Bien vos éléments sont notés : [");
for(i = 0; i < taille; ++i) {
    if(i) printf(", ");
    printf("%g", tableau[i]);
}
printf("]\n");
exit(EXIT_SUCCESS);
}

```

Exercice 87 (★★ Allocation dynamique deux dimensions naïve).

Compléter le code suivant pour de sorte de :

- Allouer un tableau dynamique d'adresses pour la première dimension.
- Pour chaque adresse allouer un tableau dynamique pour la seconde dimension.

```

#include <stdio.h>
#include <stdlib.h>

int main() {
    float ** tableau = NULL;
    int taille_first = 6, taille_second = 4;
    int i, j;
    /* TODO : allouer dynamiquement
       → tableau[taille_first][taille_second] */
    for(i = 0; i < taille_first; ++i) {
        for(j = 0; j < taille_second; ++j) {
            tableau[i][j] = (float)(i + 1) / (j + 1);
        }
    }
    for(i = 0; i < taille_first; ++i) {
        for(j = 0; j < taille_second; ++j) {
            if(j) printf(" ");
            printf("%.5f", tableau[i][j]);
        }
        printf("\n");
    }
    exit(EXIT_SUCCESS);
}

```

```
}
```

Exercice 88 (★★ Allocation dynamique deux dimensions en deux allocations).

Compléter le code suivant pour de sorte de :

- Allouer un tableau dynamique d'adresses pour la première dimension.
- Allouer un tableau dynamique pour la seconde dimension (unique).
- Faire le lien entre les deux tableaux.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    float ** tableau = NULL;
    float * valeurs = NULL;
    int taille_first = 6, taille_second = 4;
    int i, j;
    /* TODO : allouer dynamiquement
       → tableau[taille_first][taille_second] */
    /* TODO : allouer dynamiquement valeurs[taille_first *
       → taille_second] */
    /* TODO : relier les deux tableaux */
    for(i = 0; i < taille_first; ++i) {
        for(j = 0; j < taille_second; ++j) {
            tableau[i][j] = (float)(i + 1) / (j + 1);
        }
    }
    for(i = 0; i < taille_first; ++i) {
        for(j = 0; j < taille_second; ++j) {
            if(j) printf(" ");
            printf("%.5f", tableau[i][j]);
        }
        printf("\n");
    }
    exit(EXIT_SUCCESS);
}
```

Exercice 89 (★★ Allocation dynamique via une fonction).

Compléter le code suivant pour de sorte de :

- Allouer un tableau dynamique à deux dimensions via `alloc_tab`.
- Libérer le tableau dynamique via `free_tab`.

```
#include <stdio.h>
#include <stdlib.h>

int alloc_tab(float *** tableau, int taille_first, int
→ taille_second) {
    /* TODO : allouer tableau et initialiser ses valeurs à (float)(i
    → + 1) / (j + 1) */
    return 1;
}

void free_tab(float *** tableau) {
    /* TODO : libérer tableau (sans fournir ses dimensions) */
}

int main() {
    float ** tableau = NULL;
    int taille_first = 6, taille_second = 4;
    int i, j;
    if(! alloc_tab(&tableau, taille_first, taille_second)) {
        printf("Erreur allocation tableau : arrêt.\n");
        exit(EXIT_FAILURE);
    }
    for(i = 0; i < taille_first; ++i) {
        for(j = 0; j < taille_second; ++j) {
            if(j) printf(" ");
            printf("%.5f", tableau[i][j]);
        }
        printf("\n");
    }
    free_tab(&tableau);
    exit(EXIT_SUCCESS);
}
```

Exercice 90 (★★★ Statistiques sur liste d'entiers).

Écrire un programme qui lit une liste d'entiers et affiche les statistiques suivantes :

- Valeur minimale.
- Valeur maximale.
- Moyenne des valeurs.

```
Entrez des entiers positifs : 14 12 10 8 7 -1
min : 7
max : 14
moyenne : 10.2
```

Exercice 91 (★★★ Extraire de l'information formatée d'une chaîne de caractères).

Compléter le code suivant de manière à extraire les informations de la chaîne de caractère. Puis afficher ces informations :

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    const char * infos = "Linus Torvalds 52 ans C";
    /* TODO : extraire les informations */
    exit(EXIT_SUCCESS);
}
```

```
Prenom : Linus
Nom : Torvalds
Age : 52
Parle couramment la langue C
```

Exercice 92 (★★★ Mini-interpréteur sur liste d'entiers).

Un ami a entendu parler du langage Brainfuck. C'est un langage où vous pouvez vous déplacer dans la mémoire et y changer des valeurs. Il aimeraient que vous en codiez un mini-interpréteur dans une plage de taille donnée par l'utilisateur en suivant les règles suivantes :

- '+' ajoute 1 à la case mémoire regardée par le curseur.
- '-' retire 1 à la case mémoire regardée par le curseur.
- '=' égalise toutes les cases mémoire à la valeur de la case mémoire regardée.
- '>' déplace le curseur vers la droite (retour au début si dépasse de la plage mémoire).
- '<' déplace le curseur vers la gauche (retour à la fin si dépassé de la plage mémoire).
- '.' affiche le contenu de la plage mémoire.

```
taille de la mémoire : 4
.
[0, 0, 0, 0]
+.
[1, 0, 0, 0]
=.
[1, 1, 1, 1]
>+.
[1, 3, 1, 1]
=.
[3, 3, 3, 3]
<--.
[1, 3, 3, 3]
<++++.
[1, 3, 3, 7]
-->>.
[1, 2, 3, 5]
```

Exercice 93 (★★★ Jeu du Morpion).

Coder sous console un jeu de Morpion :

- Jeu à deux joueurs dans une grille de 3×3 .
- Tour par tour chacun place un pion.
- Lorsque 3 pions d'un même joueur sont alignés, il gagne.

```
+---+  
|X| |O|    Le joueur X gagne !  
+---+  
|   |X| |  
+---+  
|O| |X|  
+---+
```

Exercice 94 (★★★ Enregistrement et recherche de numéros).

Écrire un programme qui demande une liste de noms et associe à chaque nom un numéro. Une fois que l'utilisateur a validé sa liste, lui proposer de rechercher un nom. Si le nom a été renseigné, on lui affiche le numéro associé. Une sortie de ce programme pourrait ressembler à la suivante :

```
Nom (None pour arrêter) : Personne
Numéro : 42
Nom (None pour arrêter) : Moi
Numéro : 1337
Nom (None pour arrêter) : Lui
Numéro : 1234
Nom (None pour arrêter) : None
Nom à rechercher (None pour arrêter) :
>>> Personne
Le numéro de "Personne" est 42
Nom à rechercher (None pour arrêter) :
>>> Toi
"Toi" non trouvé.
Nom à rechercher (None pour arrêter) :
>>> Moi
Le numéro de "Moi" est 1337
Nom à rechercher (None pour arrêter) :
>>> None
```

Exercice 95 (★★★★★ Suite de Fibonacci généralisée et optimisée).

Oscar s'intéresse à la généralisation de la suite de Fibonacci. Il a découvert la suite de Tribonacci, la suite de Tétranacci et qu'elles semblent se généraliser. Il vous indique que les suites énoncées peuvent s'écrire pour tout entier positif n par les relations suivantes :

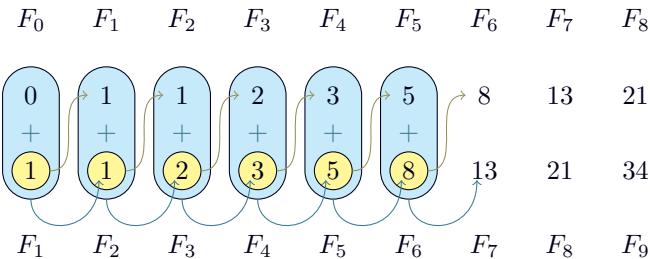
- Fibonacci (degré 2) :

$$F_n = \begin{cases} 0 & \text{Si } n = 0 \\ 1 & \text{Si } n = 1 \\ F_{n-1} + F_{n-2} & \text{Sinon} \end{cases}$$

Pour exemples :

$$\left| \begin{array}{lcl} F_2 & = & F_{2-1} + F_{2-2} \\ & = & F_1 + F_0 \\ & = & 1 + 0 \\ & = & 1 \end{array} \right| \left| \begin{array}{lcl} F_3 & = & F_{3-1} + F_{3-2} \\ & = & F_2 + F_1 \\ & = & 1 + 1 \\ & = & 2 \end{array} \right| \left| \begin{array}{lcl} F_5 & = & F_{5-1} + F_{5-2} \\ & = & F_4 + F_3 \\ & = & 3 + 2 \\ & = & 5 \end{array} \right.$$

Conceptuellement ceci se calcule depuis les deux termes précédents :



(Tourner la page pour la suite)

- Tribonacci (degré 3 : addition des 3 termes précédents) :

$$A_n = \begin{cases} 0 & \text{Si } n < 2 \\ 1 & \text{Si } n = 2 \\ A_{n-1} + A_{n-2} + A_{n-3} & \text{Sinon} \end{cases}$$

- Tétranacci (degré 4 : addition des 4 termes précédents) :

$$B_n = \begin{cases} 0 & \text{Si } n < 3 \\ 1 & \text{Si } n = 3 \\ B_{n-1} + B_{n-2} + B_{n-3} + B_{n-4} & \text{Sinon} \end{cases}$$

- Généralisation (degré d : addition des d termes précédents) :

$$F_{d,n} = \begin{cases} 0 & \text{Si } n < d - 1 \\ 1 & \text{Si } n = d - 1 \\ \underbrace{F_{d,n-1} + F_{d,n-2} + \cdots + F_{d,n-d}}_{d \text{ éléments}} & \text{Sinon} \end{cases}$$

Afficher les 20 premières valeurs de chaque suite en fonction du degré demandé. Votre méthode sera considérée optimisée si elle permet de calculer le terme d'indice 100 en temps et en mémoire raisonnables. Oscar a entendu parler du calcul de la suite de Fibonacci en temps logarithmique, mais ceci reste une légende et est probablement inscrit quelque part dans la Matrice.

```
Degré de la suite : 1
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
Degré de la suite : 2
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
→ 1597, 2584, 4181]
```

```
Degré de la suite : 3
[0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705,
→ 3136, 5768, 10609, 19513]
```

```
Degré de la suite : 4
[0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, 208, 401, 773, 1490,
→ 2872, 5536, 10671, 20569]
```

```
Degré de la suite : 5
[0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, 120, 236, 464, 912, 1793,
 ↵ 3525, 6930, 13624]
```

```
Degré de la suite : 8
[0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 64, 128, 255, 509,
 ↵ 1016, 2028]
```

Attendus :

- Fonctionnement pour la suite de Fibonacci.
- Calcul de valeurs exactes.
- Fonctionnement récursif.
- Fonctionnement optimisé.
- Calcul en temps logarithmique (s'intéresser aux matrices pour ce critère).

10 Fichiers

Nous avons vu comment communiquer avec l'utilisateur, organiser nos instructions pour réaliser un programme. Cependant, à la fin de l'exécution de notre programme, nous reviendrions à zéro en le relançant. Il serait donc intéressant d'avoir un moyen de sauvegarder cet état. De même nous nous sommes limités à des entrées courtes et simples données par un utilisateur lors de l'exécution de son programme.

Pour sauvegarder et récupérer de l'information, il est possible de passer par la gestion des fichiers sur l'espace disque.

10.1 Ouverture et création de fichiers

Les fichiers vont se gérer dans le programme à l'aide du type `FILE *`. Celui-ci permettra de garder un pointeur sur un fichier ouvert par le programme. L'ouverture d'un fichier depuis l'espace disque se fait par la commande `fopen` de `stdio.h` :

```
fopen(/* chemin du fichier */ , /* mode d'ouverture */)
```

- Le premier argument est le chemin du fichier sur l'espace disque.
- Le second argument est le mode d'ouverture du fichier, ceci dépend si l'on souhaite lire, écrire, s'y déplacer et autre :

Mode	Lecture	Écriture	Création	Effacement	Ajout en fin
"r"	✓				
"w"		✓	✓	✓	
"a"		✓	✓		✓
"r+"	✓	✓			
"w+"	✓	✓	✓	✓	
"a+"	✓	✓	✓		✓

Une fois les opérations dans le fichier terminées, on arrête son utilisation par `fclose` sur celui-ci :

```
FILE * fichier = NULL;
/* Tentative d'ouverture / création d'un fichier */
if((fichier = fopen(/* chemin */, /* mode */)) == NULL) {
    /* Gestion de l'impossibilité d'ouverture */
}
/* Opérations avec le fichier */
/* ... */
/* Fermeture du fichier */
fclose(fichier);
fichier = NULL;
```

L'exemple suivant permet la création d'un fichier `MonFichier.txt` vide :

```
FILE * fichier = NULL;
if((fichier = fopen("MonFichier.txt", "w")) == NULL) {
    printf("Erreur de création de mon fichier.\n");
    exit(EXIT_FAILURE);
}
printf("Fichier créé avec succès.\n");
fclose(fichier);
fichier = NULL;
exit(EXIT_SUCCESS);
```

Pour alléger la lecture des codes suivants, la vérification de l'existence ou création du fichier peut être omise. Cependant, pensez à le gérer pour éviter des mauvaises surprises dans vos codes.

10.2 Lecture et écriture

10.2.1 Avec fonctions formatées

Pour communiquer avec l'utilisateur via le terminal nous avons vu des fonctions issues de `stdio` qui le permettent telles que `printf`, `scanf`. Ces fonctions existent aussi en version pour les fichiers.

fprintf

`fprintf` permet l'impression formatée de caractères dans un fichier donné. Il fonctionne de la même manière que `printf`, à la différence qu'il faut fournir un premier argument supplémentaire : le pointeur sur le fichier :

```
FILE * fichier = fopen("test.txt", "w");
fprintf(fichier, "Hello fichier !\n");
fclose(fichier);
```

De même `fprintf` peut être utilisé pour sauvegarder des données fournies par à un utilisateur :

```
int renseignerInfos(const char * pseudo) {
    int age;
    printf("Quel est votre âge ? ");
    scanf("%d", &age);
    FILE * infos = fopen(pseudo, "w");
    /* sauvegarde la valeur de age */
    fprintf(infos, "%d\n", age);
    fclose(infos);
}
```

fscanf

La fonction `fscanf` fonctionne de la même manière pour récupérer le contenu formaté d'un fichier :

```
int lireInfos(const char * pseudo, int * age) {
    FILE * infos = NULL;
    if((infos = fopen(pseudo, "r")) == NULL) {
        return 0; /* l'utilisateur est inconnu */
    }
    /* récupère la valeur de age */
    fscanf(infos, "%d", age);
    fclose(infos);
    return 1;
}
```

10.2.2 Par caractère

Dans des cas plus pratiques, nous pourrions vouloir récupérer ou écrire les informations caractère par caractère. Par exemple si vous souhaitez coder un message en conservant la mise en page. Pour ceci, nous avons à disposition les fonctions `fputc` pour écrire un caractère dans un fichier et `fgetc` pour lire un caractère depuis un fichier.

fgetc

`fgetc` va lire et renvoyer un par un les caractères d'un fichier donné. Lorsque `fgetc` est arrivé à la fin du fichier, la valeur renvoyée sera `EOF` (End Of File).

```
FILE * fichier = fopen("message.txt", "r");
int caractere;
/* tant qu'on lit un caractère dans le fichier */
while((caractere = fgetc(fichier)) != EOF) {
    putchar(caractere); /* on affiche le caractère */
}
fclose(fichier);
```

fputc

Il est tout aussi possible d'écrire dans un fichier caractère par caractère. Ceci peut se faire à l'aide de `fputc`.

```
FILE * input = fopen("message.txt", "r");
FILE * output = fopen("resultat.txt", "w");
int caractere;
const int cle = 5;
/* tant qu'on lit un caractère dans le fichier */
while((caractere = fgetc(input)) != EOF) {
    /* on le code s'il est alphabétique */
    if(caractere >= 'a' && caractere <= 'z')
        caractere = (caractere - 'a' + cle) % 26 + 'a';
    else if(caractere >= 'A' && caractere <= 'Z')
```

```

    caractere = (caractere - 'A' + cle) % 26 + 'A';
/* on l'écrit dans le fichier de sortie*/
    fputc(caractere, output);
}
fclose(input);
fclose(output);

```

10.2.3 Par bloc mémoire

Pour effectuer des sauvegardes, nous pourrions souhaiter de ne pas avoir nécessité que le fichier soit humainement lisible et ne prenne pas une place plus importante que celle des données réelles. Pour ceci, il est possible d'écrire dans un fichier en binaire directement.

Cette écriture en binaire se fera par octets. On lui passera un tableau ou un pointeur contenant ou recevant les données qui nous intéressent. On précisera que le fichier manipulé n'est pas un fichier texte en ajoutant un "b" après le mode d'ouverture "r", "w" ou "a". Les fonctions utilisées ensuite pour la lecture et l'écrire seront **fread** et **fwrite**.

fwrite

La fonction **fwrite** prend en paramètres :

- Pointeur sur les données à écrire.
- Taille d'un élément.
- Nombre d'éléments.
- Pointeur sur le fichier où écrire.

fwrite renvoie le nombre d'éléments écrits dans le fichier.

```

int sauvegarderListe(const char * filepath, const int * liste, int
→ taille) {
    FILE * output = fopen(filepath, "wb");
/* écriture de la taille : une variable */
    if(fwrite(&taille, sizeof(int), 1, output) != 1) {
        printf("Erreur écriture taille\n");
        return 0;
    }
}

```

```

/* écriture de la liste : un tableau */
if(fwrite(liste, sizeof(int), taille, output) != taille) {
    printf("Erreur écriture liste\n");
    return 0;
}
fclose(output);
return 1;
}

```

fread

Une liste ainsi écrite précédemment peut être charger par le même type de procédé. **fread** permet la lecture de données et fonctionne similairement à **fwrite**. À noter que **fread** ne procède pas à l'allocation de la plage mémoire dans laquelle elle écrit. Ce sera votre rôle de vous assurer qu'elle existe et peut recevoir les données.

La fonction **fread** prend en paramètres :

- Pointeur sur les données à récupérer.
- Taille d'un élément.
- Nombre d'éléments.
- Pointeur sur le fichier où lire.

fread renvoie le nombre d'éléments lus depuis le fichier.

```

int chargerListe(const char * filepath, int ** liste, int *
→ taille) {
    FILE * input = fopen(filepath, "rb");
/* lecture de la taille : nécessaire à l'allocation */
if(fread(taille, sizeof(int), 1, input) != 1) {
    printf("Erreur lecture taille\n");
    return 0;
}
/* allocation de la liste */
if((*liste = (int *)malloc(sizeof(int) * *taille)) == NULL) {
    printf("Erreur allocation liste\n");
}

```

```
    return 0;
}

/* lecture des éléments de la liste */
if(fread(*liste, sizeof(int), *taille, input) != *taille) {
    printf("Erreur lecture liste\n");
    return 0;
}
fclose(input);
return 1;
}
```

10.3 Se déplacer dans un fichier

Des fonctions sont proposées pour jouer sur la position du curseur lors de l'écriture et de la lecture dans un fichier.

`ftell` prend pour argument un fichier et indique la position actuelle du curseur dans ce fichier.

`rewind` prend pour argument un fichier et rembobine le fichier au début.

`fseek` permet de placer le curseur à un endroit souhaité dans un fichier. `fseek` prend en arguments :

- Le fichier dans lequel déplacer le curseur.
- La position relative où déplacer le curseur par rapport à l'information donnée au paramètre suivant.
- Point de repère depuis lequel appliquer le décalage :
 - `SEEK_SET` : début du fichier.
 - `SEEK_CUR` : position actuelle dans le fichier.
 - `SEEK_END` : fin du fichier.

En exemple d'application de ces fonctions, nous proposons un code qui lit le texte pour compter le nombre de phrases présentes puis lit chaque phrase :

```
int carInChaine(char car, const char * chaine) {
    for(; *chaine != '\0'; ++chaine) {
```

```
    if(car == *chaine)
        return 1;
    }
    return 0;
}

int lirePhrase(FILE * file, long * start, long * end) {
    int car;
    while((car = fgetc(file)) != EOF) {
        if(! carInChaine(car, "\t\n")) {
            break;
        }
    }
    if(start) /* on récupère la position du début de la phrase */
        *start = ftell(file) - 1;
    do {
        if(car == '.') {
            break;
        }
    } while((car = fgetc(file)) != EOF);
    if(end) /* on récupère la position de la fin de la phrase */
        *end = ftell(file);
    return car != EOF;
}

void afficherPortionFichier(FILE * file, long start, long end) {
    int car;
    /* on se replace dans le fichier à la position indiquée */
    fseek(file, start, SEEK_SET);
    while(ftell(file) != end) {
        putchar(fgetc(file));
    }
}
```

```
int main() {
    FILE * file = fopen("message.txt", "r");
    int i;
    long start, end;
    for(i = 0; lirePhrase(file, NULL, NULL); ++i);
    printf("%d phrases.\n", i);
    /* on rembobine le fichier au début */
    rewind(file);
    for(i = 0; lirePhrase(file, &start, &end); ++i) {
        printf(" - Phrase %d (%ld caractères): ", i + 1, end - start);
        afficherPortionFichier(file, start, end);
        printf("\n");
    }
    fclose(file);

    exit(EXIT_SUCCESS);
}
```

10.4 Flux standards d'entrées et sorties

En réalité, lorsque vous utilisez `printf` et `scanf`, vous travaillez également dans des `FILE *` standards qui correspondent aux entrées et sorties de votre terminal.

10.4.1 `stdout`

Le flux de sortie dans lequel on imprime nos caractères lors des appels à `printf` par exemple est `stdout`. On peut l'utiliser avec `fprintf` :

```
printf("Par printf\n");
fprintf(stdout, "Par fprintf\n");
```

10.4.2 `stdin`

De la même manière il existe le flux de sortie standard `stdin` peut être utilisé avec `fscanf` :

```

int nombreScanf, nombreFscanf;
scanf("%d", &nombreScanf);
fscanf(stdin, "%d", &nombreFscanf);
printf("%d %d\n", nombreScanf, nombreFscanf);

```

10.4.3 stderr

Le flux standard qui sera cependant le plus utile est `stderr`. C'est un flux standard de sortie dédié à l'écriture des erreurs ou des logs. À noter que `stdout` ne sera souvent réellement imprimé dans le terminal que lorsque son buffer est plein ou lorsqu'il imprime un retour à la ligne. `stderr` sera imprimé peu importe les caractères donnés.

```

FILE * fichier = NULL;
if((fichier = fopen("fichier_a_ne_pas_creer", "r")) == NULL) {
    fprintf(stderr, "Erreur main() : \"fichier_a_ne_pas_creer\" est
    ↪ introuvable\n");
    exit(EXIT_FAILURE);
}
fclose(fichier);

```

À noter que cette sortie peut être redirigée vers un fichier de logs, laissant dans le terminal uniquement `stdout` :

```

# gcc -o prog main.c
# ./prog
Erreur main() : "fichier_a_ne_pas_creer" est introuvable
# ./prog 2>log
# cat log
Erreur main() : "fichier_a_ne_pas_creer" est introuvable

```

10.5 Résumé

Il est possible de gérer un fichier avec le type FILE *. Ce fichier s'ouvre avec `fopen` et se ferme avec `fclose` :

```
FILE * fichier = NULL;
if((fichier = fopen(/* chemin */, /* mode */)) == NULL) {
    /* traitement erreur */
}
/* utilisation fichier */
fclose(fichier);
```

Les principaux modes sont :

- **r** : lecture.
- **w** : écriture (création et effacement du fichier).
- **a** : écriture (création et ajout en fin du fichier).
- **r+** : écriture et lecture.
- **w+** : écriture et lecture (création et effacement du fichier).
- **a+** : écriture et lecture (création et ajout en fin du fichier).

Il est possible d'imprimer une chaîne de caractères formatée dans un fichier avec `fprintf` et lire une chaîne de caractères formatée avec `fscanf` :

```
fprintf(fichier, /* format */, /* variables */);
scanf(fichier, /* format */, /* adresses */);
```

Il est aussi possible de procéder caractère par caractère avec `fgetc` et `fputc` :

```
char car;
while((car = fgetc(fichier)) != EOF) {
    fputc(car, fichier);
}
```

De manière plus avancée, il est possible de passer en binaire en ajoutant **b** au mode. Ceci permet d'écrire nos données avec `fwrite` et lire avec `fread` :

```
if(fwrite(/* pointeur */, /* taille élément */, /* nombre
→ éléments */, fichier) != /* nombre éléments */)
    /* gestion erreur écriture */
if(fread(/* pointeur */, /* taille élément */, /* nombre éléments
→ */, fichier) != /* nombre éléments */)
    /* gestion erreur lecture */
```

Il est possible de jouer avec la position du curseur dans un fichier avec les fonctions suivantes :

```
ftell(fichier); /* donne la position du curseur */
rewind(fichier); /* met le curseur au début du fichier */
fseek(fichier, 1, SEEK_SET); /* se positionne après le premier
→ octet du fichier */
fseek(fichier, 1, SEEK_CUR); /* déplace le curseur d'un caractère
→ vers l'avant */
fseek(fichier, -1, SEEK_END); /* se positionne avant le dernier
→ octet du fichier */
```

La bibliothèque `stdio` définit des entrées en sorties standards :

```
stdout /* sortie standard, utilisée par printf */
stdin /* entrée standard, utilisée par scanf */
stderr /* sortie d'erreur standard */
```

10.6 Entraînement

Exercice 96 (★ Création d'un fichier).

Compléter le code suivant pour de sorte de :

- Ouvrir un fichier.
- Écrire dans le fichier.
- Ferme le fichier.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    const char * chemin = "test.txt";
    /* TODO : Créer le fichier au chemin donné et y écrire "Test.\n"
    ↵ */
    exit(EXIT_SUCCESS);
}
```

Exercice 97 (★★ Lire des fichiers).

Compléter le code suivant pour de sorte de :

- Ouvrir chaque fichier listé en argument.
- Écrire son contenu dans le terminal.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
    FILE * output = stdout;
    /* TODO : écrire le contenu des fichiers listés en arguments
    ↵ dans output */
    exit(EXIT_SUCCESS);
}
```

Exercice 98 (★★ Fichier en binaire).

Compléter le code suivant pour de sorte de :

- Ouvrir le fichier en lecture s'il existe.
- Lire un entier en binaire.
- Effectuer un traitement.
- Ouvrir le fichier en écriture.
- Écrire un entier en binaire.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    const char * path = "nombre.bin";
    int compteur = 0;
    /* TODO : compteur le nombre de fois où le programme est lancé
    ↵ */
    /* TODO : maintenir ce compteur en écriture avec fread et fwrite
    ↵ */
    printf("Programme lancé %d fois\n", compteur);
    exit(EXIT_SUCCESS);
}
```

Exercice 99 (★★ Éditer un fichier).

Compléter le code suivant pour de sorte de :

- Ouvrir un fichier et changer la case de son contenu.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    const char * path = "exemple.txt";
    FILE * fichier = NULL;
    /* TODO : mettre en majuscules le texte au chemin donné en
    ↵ n'ouvrant qu'un seul fichier */
    exit(EXIT_SUCCESS);
}
```

Exercice 100 (★★ Compteur de lancements).

Écrire un programme qui compte le nombre de fois où il a été lancé. Ceci pourrait se faire à l'aide de la sauvegarde d'un entier dans un fichier. Un utilisateur doit pouvoir ouvrir le fichier en clair.

```
# ./prog
Programme lancé 1 fois
# ./prog
Programme lancé 2 fois
# ./prog
Programme lancé 3 fois
# ./prog
Programme lancé 4 fois
```

Exercice 101 (★★★ Codage Vigenère depuis fichier).

Écrire un programme qui lit du texte depuis un fichier texte. Puis l'encode ou le décode avec la méthode du chiffre de Vigenère pour l'afficher sans la sortie standard.

```
# ./prog
Attendu : ./prog [FICHIER MESSAGE] [CLE]
Attendu : ./prog [FICHIER MESSAGE] [CLE] decode
# ./prog message.txt ESGI
Ipkutdk li lkfxw zzsh ipsmkbwx !
Kvjat dsarà, çi hwbzeaz ti xgqvw.
```

Exercice 102 (★★ Sauvegarde et chargement en binaire).

Écrire un programme qui sauvegarde et charge un personnage dans un fichier binaire. La manipulation du programme se fait en ligne de commande en passant des options au programme. L'écriture dans le fichier doit respecter la spécification suivante :

- (4 octets) entier : nombre de caractères constituant le nom du personnage.
- (N octets) chaîne de caractères : nom du personnage constitué de N caractères.
- (4 octets) entier : statistique de vie du personnage.
- (4 octets) entier : statistique d'attaque du personnage.
- (4 octets) entier : statistique de défense du personnage.
- (4 octets) entier : statistique de vitesse du personnage.

```
# ./prog
Attendu :
    ./prog -create [FICHIER] [NOM] [VIE] [ATK] [DEF] [VIT]
    ./prog -read [FICHIER]
# ./prog -create chouette.perso "Chouette Oiseau" 127 42 87 94
# ./prog -read chouette.perso
Personnage :
    Nom : Chouette Oiseau
    Vie : 127
    Attaque : 42
    Défense : 87
    Vitesse : 94
}
```

```
# hexedit chouette.perso
00000000  0F 00 00 00  43 68 6F 75  65 74 74 65  ....Chouette
0000000C  20 4F 69 73  65 61 75 7F  00 00 00 2A  Oiseau....*
00000018  00 00 00 57  00 00 00 5E  00 00 00      ...W...^...
00000024
--- chouette.perso --0x0/0x23-----
```

Exercice 103 (★★★ Enregistrement et recherche de numéros avec sauvegarde).

Écrire un programme qui :

- prend une option `-i [FICHIER]` pour ouvrir une liste de noms / numéros depuis un fichier existant.
- prend une option `-o [FICHIER]` pour enregistrer une liste de noms / numéros depuis la liste actuellement connue par le programme.
- Ouvre si elle existe une liste de noms indiquée.
- Propose à l'utilisateur l'ajout de nouvelles associations noms / numéros.
- Enregistre les associations dans une liste de noms indiquée.
- Permet la recherche d'un numéro depuis le nom associé.

```
# ./prog -o liste.txt
Nom (None pour arrêter) : Premier 1
Numéro : Nom (None pour arrêter) : Second 2
Numéro : Nom (None pour arrêter) : None
Nom à rechercher (None pour arrêter) :
>>> None
# ./prog -i liste.txt -o liste.txt
Nom (None pour arrêter) : Troisième 3
Numéro : Nom (None pour arrêter) : None
Nom à rechercher (None pour arrêter) :
>>> Premier
Le numéro de "Premier" est 1
Nom à rechercher (None pour arrêter) :
>>> Second
Le numéro de "Second" est 2
Nom à rechercher (None pour arrêter) :
>>> Troisième
Le numéro de "Troisième" est 3
Nom à rechercher (None pour arrêter) :
>>> None
```

11 Structures

Dans notre code, nous embarquons souvent plusieurs informations qui correspondent en réalité à une même entité et qu'il faut passer à nos fonctions. Une alternative peu propre et qui peut vite poser problème lorsque nous souhaitons réutiliser une fonction est d'avoir ces données en variables globales. Nous nous intéresserons ici à organiser nos informations dans notre code pour plus de maintenabilité.

11.1 Typedef

Une manière de créer un synonyme d'un type est l'opérateur `typedef`. Il peut être pratique lorsqu'on fera appel régulièrement à un type long ou compliqué à écrire. Il est par exemple possible d'abréger l'appel de `unsigned int` en `uint` :

```
typedef unsigned int uint;

int main() {

    uint entierPositif = 4000000000;
    printf("%u\n", entierPositif);

    exit(EXIT_SUCCESS);
}
```

Il est possible de complexifier le type sur lequel on veut faire un alias, comme un tableau ou un pointeur :

```
typedef int intListeStatique[];
/* intListeStatique sera équivalent au type d'un tableau de int
   */
typedef int * intListe;
/* intListe sera équivalent au type d'un pointeur sur un int */
```

```

/* équivalent à avoir "int * liste" en argument */
void afficherIntListe(intListe liste) {
    int i;
    for(i = 0; liste[i] >= 0; ++i) {
        if(i) printf(" , ");
        printf("%d", liste[i]);
    }
    printf("\n");
}

int main() {
    /* équivalent à "int liste[] = {1, 2, 3, 4, -1};" */
    intListeStatique liste = {1, 2, 3, 4, -1};
    afficherIntListe(liste);

    exit(EXIT_SUCCESS);
}

```

11.2 Structures

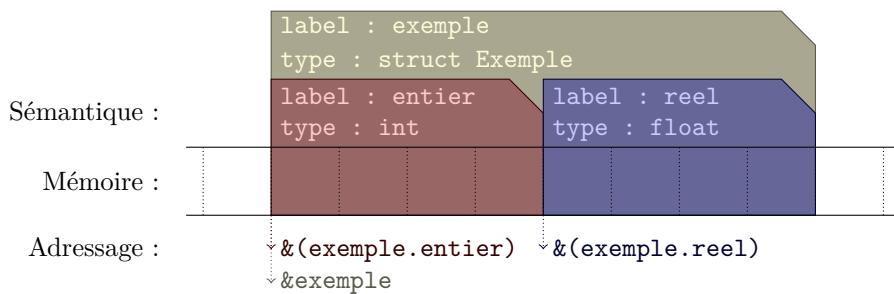
En langage C, un structure correspondra à une entité qui permettra d'en regrouper plusieurs. Par exemple, lorsque vous avez une liste, il faudra potentiellement avoir à disposition la liste, sa taille, une capacité maximale et autres. Il est possible de modéliser cette liste par un élément qu'est la structure et ne passer que cet élément aux fonctions qui l'auraient en paramètres.

```

struct Exemple {
    int entier;
    float reel;
};

struct Exemple exemple;

```



11.3 Définition

Une structure se fabrique depuis le mot clé **struct**. On indique ensuite lors de sa définition les champs qui la composent. L'accès aux éléments de la structure se fera ensuite en séparant la variable fabriquée depuis la structure et le champs souhaité par un point ..

```
/* Schéma de construction d'une Liste */
struct Liste {
    int * elements; /* valeurs de la liste */
    int taille; /* taille de la liste */
};

int main() {
    /* Construction d'une liste */
    struct Liste liste;
    /* accès aux champs de la liste */
    liste.elements;
    liste.taille;

    exit(EXIT_SUCCESS);
}
```

Comme vu précédemment, il est possible de fabriquer un alias du même nom pour s'éviter l'écriture de **struct** à la construction d'une liste juste avant sa définition.

```

/* Alias */
typedef struct Liste Liste;
/* Schéma de construction d'une Liste */
struct Liste {
    int * elements; /* valeurs de la liste */
    int taille; /* taille de la liste */
};

int main() {
    /* Construction d'une liste */
    Liste liste;
    /* accès aux champs de la liste */
    liste.elements;
    liste.taille;

    exit(EXIT_SUCCESS);
}

```

11.4 Avec des pointeurs

À noter que souvent en langage C, les structures seront souvent gérées par pointeurs. L'accès aux champs d'un pointeur sur une structure peut se faire principalement de deux manière équivalentes :

```

Liste * liste;
/* déréférencement puis accès */
(*liste).elements;
/* version raccourcie */
liste->elements;

```

Il sera souvent plus pratique d'avoir des fonctions qui permettent l'allocation et la libération d'une structure. Ceci en particulier si elle est allouée dynamiquement ou que ses champs peuvent l'être :

```
Liste * Liste_alloc(int taille) {
    Liste * res = NULL;
    if((res = (Liste *)malloc(sizeof(Liste))) == NULL) {
        fprintf(stderr, "Liste_alloc : Erreur alloc liste\n");
        return NULL;
    }
    res->taille = taille;
    if((res->elements = (int *)calloc(taille, sizeof(int))) == NULL)
        {
            free(res);
            fprintf(stderr, "Liste_alloc : Erreur alloc éléments\n");
            return NULL;
        }
    return res;
}

void Liste_free(Liste ** liste) {
    free((*liste)->elements);
    free(*liste);
    *liste = NULL;
}

int main() {
    /* Construction d'un pointeur de liste */
    Liste * liste = Liste_alloc(4);
    /* accès aux champs de la liste référencée */
    liste->elements;
    liste->taille;

    Liste_free(&liste);

    exit(EXIT_SUCCESS);
}
```

Il est aussi possible de masquer l'aspect pointeur à l'utilisateur en le précisant

dans le `typedef`. À noter que si vous souhaitez aller plus loin et respecter les principes d'encapsulation que l'on peut voir dans d'autres langages, vous pouvez déclarer l'existence de la structure à l'avance. Notez que vous ne pourrez l'utiliser dans le code qu'après sa définition ou présenter un pointeur avant (nécessité pour le compilateur de connaître la taille de la structure lorsqu'il opère réellement dessus là où la taille d'une adresse est fixe).

```
/* Alias version pointeur */
typedef struct Liste * Liste;
/* Déclaration en amont */
struct Liste;

Liste Liste_alloc(int taille);

void Liste_free(Liste * liste);

int main() {

    /* Construction d'un pointeur de liste */
    Liste liste = Liste_alloc(4);
    /* La liste ne peut plus être manipulée directement */
    /* Il faut maintenant passer par des fonctions */
    Liste_free(&liste);

    exit(EXIT_SUCCESS);
}

/* Votre partie du programme : */

/* pourra être cachée à l'utilisateur de votre code */
/* Schéma de construction d'une Liste */
struct Liste {
    int * elements; /* valeurs de la liste */
    int taille; /* taille de la liste */
};
```

```

Liste Liste_alloc(int taille) {
    Liste res = NULL;
    if((res = (Liste)malloc(sizeof(struct Liste))) == NULL) {
        fprintf(stderr, "Liste_alloc : Erreur alloc liste\n");
        return NULL;
    }
    res->taille = taille;
    if((res->elements = (int *)calloc(taille, sizeof(int))) == NULL)
        {
        free(res);
        fprintf(stderr, "Liste_alloc : Erreur alloc éléments\n");
        return NULL;
    }
    return res;
}

void Liste_free(Liste * liste) {
    free((*liste)->elements);
    free(*liste);
    *liste = NULL;
}

```

11.4.1 Champs de bits

Une optimisation mémoire existe pour les structures. En effet, il est possible de préciser le nombre de bits sur lequel un champ doit être stocké. Ceci se fait en précisant après le champs : puis le nombre de bits nécessaires au maximum pour le champ. Ceci peut être intéressant si la structure contient beaucoup de booléens par exemple et qu'elle est gardée un nombre important de fois en mémoire.

```

struct Personnage {
    unsigned int pointsDeVie;
    unsigned int pointsDeVieMax;
    unsigned int niveau;
    unsigned long experience;
}

```

```
unsigned char aStatutPoison;
unsigned char aStatutParalyse;
unsigned char aStatutEndormi;
unsigned int attaque;
unsigned int defense;
unsigned int attaqueSpe;
unsigned int defenseSpe;
unsigned int vitesse;
};

struct PersonnageCompress {
    unsigned int pointsDeVie : 10;
    unsigned int pointsDeVieMax : 10;
    unsigned int niveau : 7;
    unsigned long experience : 40;
    unsigned char aStatutPoison : 1;
    unsigned char aStatutParalyse : 1;
    unsigned char aStatutEndormi : 1;
    unsigned int attaque : 10;
    unsigned int defense : 10;
    unsigned int attaqueSpe : 10;
    unsigned int defenseSpe : 10;
    unsigned int vitesse : 10;
};

int main() {
    printf("%lu\n", sizeof(struct Personnage));
    printf("%lu\n", sizeof(struct PersonnageCompress));
    exit(EXIT_SUCCESS);
}
```

48
24

11.5 Unions

Dans une construction similaire à celle des structures, il existe les `union`. À noter qu'une `union` liste des champs qui sont des alias vers une même donnée. C'est-à-dire que si une `union` regroupe un `float` et un `int`, ces deux champs pointent vers le même emplacement mémoire, mais c'est l'accès au champ qui permet de déterminer comment retranscrire l'information en le type demandé.

```
union Scalaire {
    int entier;
    float flottant;
    double flottantPrecis;
};

int main() {
    union Scalaire nombre;
    nombre.entier = 42;
    printf("sizeof(Scalaire) : %lu\n", sizeof(union Scalaire));
    printf("entier :          %d\n", nombre.entier);
    printf("flottant :        %g\n", nombre.flottant);
    printf("flottantPrecis : %g\n", nombre.flottantPrecis);
    nombre.flottant = 42;
    printf("entier :          %d\n", nombre.entier);
    printf("flottant :        %g\n", nombre.flottant);
    printf("flottantPrecis : %g\n", nombre.flottantPrecis);
    nombre.flottantPrecis = 42;
    printf("entier :          %d\n", nombre.entier);
    printf("flottant :        %g\n", nombre.flottant);
    printf("flottantPrecis : %g\n", nombre.flottantPrecis);
    exit(EXIT_SUCCESS);
}
```

Il est possible de créer des structures à l'intérieur de l'union pour que l'information codée dans l'union correspondent à plusieurs champs :

```
union Scalaire {
    int entier;
```

```

float flottant;
double flottantPrecis;
struct {
    float x;
    float y;
};

int main() {
    union Scalaire nombre;
    nombre.x = 42;
    nombre.y = 1337;
    printf("sizeof(Scalaire) : %lu\n", sizeof(union Scalaire));
    printf("entier :          %d\n", nombre.entier);
    printf("flottant :         %g\n", nombre.flottant);
    printf("flottantPrecis :  %g\n", nombre.flottantPrecis);
    printf("(x, y) :          (%g, %g)\n", nombre.x, nombre.y);
    exit(EXIT_SUCCESS);
}

```

11.6 Énumérations

Souvent pour ajouter de la sémantique et de la lisibilité au code, il peut être utile de nommer des constantes. De même, on peut souhaiter considérer qu'une variable ne devrait prendre que certaines valeurs constantes prédéfinies. Il est possible de construire un type dont les valeurs attendues seront celles de noms prédéfinis à l'aide du mot clé `enum` :

```

enum MapItem {
    MAP_VIDE,
    MAP_JOUEUR,
    MAP_ADVERSAIRE,
    MAP_MUR,
    MAP_SORTIE
};

```

```
enum MapItem item;
```

Il est possible de se passer du mot clé `enum` lors de la déclaration d'une variable de ce type énuméré à l'aide d'un `typedef` :

```
typedef enum {
    MAP_VIDE,
    MAP_JOUEUR,
    MAP_ADVERSAIRE,
    MAP_MUR,
    MAP_SORTIE
} MapItem;

MapItem item;
```

Ces types énumérés conviennent parfaitement à une utilisation dans un `switch`. À noter qu'il est recommandé d'ajouter l'option `-Wall` lors de la compilation. Si un élément de type énuméré n'est pas géré par le `switch` ceci sera précisé à la compilation.

```
typedef enum {
    ITEM_MOB_DYNAMIC,
    ITEM_MOB_STATIC,
    ITEM_MOB_UNKNOWN
} ItemMobility;

ItemMobility getMapItemMobility(MapItem item) {
    switch(item) {
        case MAP_JOUEUR :
        case MAP_ADVERSAIRE :
            return ITEM_MOB_DYNAMIC;

        case MAP_MUR :
        case MAP_SORTIE :
            return ITEM_MOB_STATIC;
    }
}
```

```

    default :
        return ITEM_MOB_UNKNOWN;
    }
}

```

À noter qu'en réalité un type énuméré est un entier dont certaines valeurs sont associées à un nom pour s'accorder avec l'utilisation qui doit en être faite. Il est possible dans l'énumération d'encoder certaines valeurs. Les valeurs successives suivront de un en un.

```

typedef enum {
    MAP_VIDE = 0,
    MAP_JOUEUR = 10,
    MAP_ADVERSAIRE,
    MAP_MUR = 20,
    MAP_SORTIE
} MapItem;

```

Puisque un type énuméré correspond à un entier et qu'un caractère aussi, il est possible de faire correspondre les valeurs du type énuméré à des caractères. Ceci peut apporter de la lisibilité dans le code et aussi avoir une conversion directe si cela devait être éditabile dans un fichier par exemple.

```

typedef enum {
    MAP_VIDE =      ' ',
    MAP_JOUEUR =    '@',
    MAP_MUR =       '#',
    MAP_ADVERSAIRE = '¤',
    MAP_SORTIE =     'x'
} MapItem;

```

11.7 Résumé

L'opérateur `typedef` permet de créer un synonyme d'un type :

```
typedef type synonyme;
```

Il est possible de créer des types structurés regroupant des champs sous un même nom :

```
struct Nom {  
    typeChamp nomChamp;  
};  
  
struct Nom variable;  
struct Nom * pointeur = &variable;  
variable.nomChamp; /* accès au champ */  
pointeur->nomChamp; /* accès au champ via pointeur */
```

De la même manière, on peut regrouper des champs sous un même nom mais que ceux-ci correspondent à un même emplacement mémoire pour réduire la taille de ce regroupement en mémoire.

```
union Nom {  
    int entier;  
    float reel;  
};  
  
union Nom nombre;  
nombre.entier = 42; /* nombre.reel est aussi modifié */  
nombre.reel = 13.37; /* nombre.entier est aussi modifié */
```

Lorsqu'un entier prend des valeurs que l'on veut nommer, on peut construire un type énuméré :

```
typedef enum {  
    VALEUR1 = 0x1,  
    VALEUR2 = 0x2  
} Liste;  
Liste liste = VALEUR1;
```

11.8 Entrainement

Exercice 104 (★ Créer une structure).

Compléter le code suivant pour de sorte de :

- Définir le modèle de la structure.
- Instancier une structure.
- Affecter des valeurs aux champs de la structure.
- Afficher les champs de la structure.

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : créer une structure Stats avec les champs entiers vie,
   ↳ attaque, defense, vitesse */

int main() {
    /* TODO : créer une structure Stats nommée perso */
    /* TODO : mettre la vie de perso à 100 */
    /* TODO : mettre l'attaque de perso à 50 */
    /* TODO : mettre la défense de perso à 70 */
    /* TODO : mettre la vitesse de perso à 30 */
    printf("Perso : {\n");
    printf("\tvie : %d\n", /* TODO : vie de perso */);
    printf("\tattaque : %d\n", /* TODO : attaque de perso */);
    printf("\tdefense : %d\n", /* TODO : defense de perso */);
    printf("\tvitesse : %d\n", /* TODO : vitesse de perso */);
    printf("}\n");
    exit(EXIT_SUCCESS);
}
```

Exercice 105 (★★ Passage par copie).

Compléter le code suivant pour de sorte de :

- Affecter une structure via une fonction et la renvoyer.
- Recevoir une structure et l'afficher.

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : utiliser votre structure Stats */

Stats Stats_creer(int vie, int attaque, int defense, int vitesse)
{
    Stats perso;
    /* TODO : renvoyer perso avec les valeurs adéquates */
    return perso;
}

void Stats_afficher(const Stats perso) {
    printf("Perso : {\n");
    /* TODO : afficher perso */
    printf("}\n");
}

int main() {
    Stats perso = Stats_creer(100, 50, 70, 30);
    Stats_afficher(perso);
    exit(EXIT_SUCCESS);
}
```

Exercice 106 (★★ Passage par adresse).

Compléter le code suivant pour de sorte de :

- Affecter une structure depuis son adresse via une fonction.
- Recevoir une structure et l'afficher depuis son adresse.

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : utiliser votre structure Stats */

int Stats_creer(Stats * perso, int vie, int attaque, int defense,
→ int vitesse) {
    if(vie < 0 || attaque < 0 || defense < 0 || vitesse < 0) {
        fprintf(stderr, "Les statistiques ne peuvent pas être
→ négatives.\n");
        return 0;
    }
    /* TODO : affecter perso avec les valeurs adéquates */
    return 1;
}

void Stats_afficher(const Stats * perso) {
    printf("Perso : {\n");
    /* TODO : afficher perso */
    printf("}\n");
}

int main() {
    Stats perso;
    if(! Stats_creer(&perso, 100, 50, 70, 30)) {
        fprintf(stderr, "Erreur création personnage : arrêt.\n");
        exit(EXIT_FAILURE);
    }
    Stats_afficher(&perso);
    exit(EXIT_SUCCESS);
}
```

Exercice 107 (★ Manipulation dynamique).

Compléter le code suivant pour de sorte de :

- Allouer et renvoyer un structure via `Stats_creer`.
- Libérer via `Stats_free` une structure allouée via `Stats_creer`.
- Afficher la structure allouée.

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : utiliser votre structure Stats */

Stats * Stats_creer(int vie, int attaque, int defense, int
→ vitesse) {
    /* TODO : allouer et affecter une structure Stats */
}

void Stats_free(Stats ** perso) {
    /* TODO : libérer une struture Stats */
}

void Stats_afficher(const Stats * perso) {
    printf("Perso : {\n");
    /* TODO : afficher perso */
    printf("}\n");
}

int main() {
    Stats * perso = NULL;
    if((perso = Stats_creer(100, 50, 70, 30)) == NULL) {
        fprintf(stderr, "Erreur création personnage : arrêt.\n");
        exit(EXIT_FAILURE);
    }
    Stats_afficher(perso);
    Stats_free(&perso);
    exit(EXIT_SUCCESS);
}
```

Exercice 108 (★★ Définir une structure Vecteur2d).

Définir une structure **Vecteur2d** avec les champs suivants :

- **double x.**
- **double y.**

Puis définir des fonctions qui permettent la manipulation d'un élément $V = (V_x, V_y)$:

- **Translation** de vecteur $T = (T_x, T_y)$:

$$V = V + T$$

- **Agrandissement** de rapport α et de centre $C = (C_x, C_y)$:

$$V = \alpha(V - C) + C$$

- **Rotation** d'angle δ et de centre $C = (C_x, C_y)$:

$$V = \begin{pmatrix} \cos(\delta) & -\sin(\delta) \\ \sin(\delta) & \cos(\delta) \end{pmatrix} (V - C) + C$$

```

Vecteur2d : (0, 0)
Translation par un Vecteur2d : (1, 2)
(1, 2)
Agrandissement de rapport 0.5 et de centre un Vecteur2d : (1, 0)
(1, 1)
Rotation d'angle 135 deg et de centre un Vecteur2d : (0, 2)
(1.11022e-16, 3.41421)

```

Exercice 109 (★★★ Structure pour gérer une grille).

Compléter le code suivant pour qu'il affiche une grille à l'écran :

```
#include <stdio.h>
#include <stdlib.h>
#include <ncurses.h>

typedef struct Grille Grille;
struct Grille {
    char * grille;
    int largeur;
    int hauteur;
};

/* donne un pointeur sur une case de la grille */
char * Grille_case(const Grille * grille, int x, int y);

/* crée une grille de taille donnée */
Grille Grille_creer(int largeur, int hauteur);

/* affiche une grille à l'écran */
void Grille_afficher(const Grille * grille);

/* libère une grille */
void Grille_free(Grille * grille);

int main() {
    int largeur = 60, hauteur = 20;
    Grille grille = Grille_creer(largeur, hauteur);
    int x = 1, y = 1;
    initscr();
    noecho();
    cbreak();
    do {
        clear();
        Grille_afficher(&grille);
        mvprintw(y, x, "@");
        mvprintw(y, x, " ");
        refresh();
        getch();
        /* gestion des événements */
    }
}
```

```
} while(1);
refresh();
clrtoeol();
refresh();
endwin();
Grille_free(&grille);
exit(EXIT_SUCCESS);
}
```

Exercice 110 (★★★ Gérer un combat de personnages).

Permettre à deux personnages de s'affronter. On vous demande de mettre en place :

- Des **Stats** contenant la vie, l'attaque, la défense et la vitesse.
- Énumération des stats.
- Des **Personnages** symbolisés par un nom, des statistiques de base et des statistiques actuelles.
- Énumération des sorts (qui influent sur les stats du joueur et ceux de l'adversaire).

Voici l'interface de combat à proposer :

```
Joueur VS Adversaire
>>>>>>>>>>>> | <<<<<<<<<<<<<<<
Vie :          100 | Vie :          100
Attaque :      50  | Attaque :      50
Defense :     20  | Defense :     20
Vitesse :     100 | Vitesse :     100
-----+-----
1 . coup      5 . soin
2 . reduire attaque 6 . augmenter attaque
3 . reduire defense 7 . augmenter defense
4 . reduire vitesse 8 . augmenter vitesse
-----+-----
Votre sort :
```

Exercice 111 (★★★ Implémenter une liste chaînée).

Un informaticien vous voit utiliser uniquement des tableaux pour vos liste. Il vous informe qu'il préfère utiliser des liste chaînées dans certains cas. Il vous propose d'implémenter les fonctionnalités d'une liste chaînée et d'une liste depuis un tableau pour vous en faire une idée vous-même. Vous préciserez pour chaque fonction sa complexité algorithmique en fonction de la taille de la liste ($\mathcal{O}(1)$ pour constante lorsque ça ne dépend pas de la taille de la liste et $\mathcal{O}(n)$ pour linéaire lorsqu'on pourrait au pire des cas passer sur chaque élément de la liste).

```
typedef struct LinkedList * LinkedList;
struct LinkedList {
    int value;
    LinkedList next;
};

/* Renvoie une liste vide */
LinkedList LL_empty();

/* Libère la liste */
void LL_free(LinkedList * liste);

/* Ajoute un élément en fin */
int LL_add_tail(LinkedList * liste, int value);

/* Ajoute un élément en tête */
int LL_add_head(LinkedList * liste, int value);

/* Ajoute un élément à une position donnée */
int LL_insert(LinkedList * liste, int id, int value);

/* Supprime l'élément en fin */
int LL_pop_tail(LinkedList * liste, int * value);

/* Supprime l'élément en tête */
int LL_pop_head(LinkedList * liste, int * value);

/* Supprime l'élément à une position donnée */
int LL_delete(LinkedList * liste, int id, int * value);

/* Affiche la liste */
void LL_print(FILE * flow, const LinkedList * liste);
```

```
typedef struct ArrayList ArrayList;
struct ArrayList {
    int * values;
    int size;
    int capacite;
};

/* Renvoie une liste vide */
ArrayList AL_empty();

/* Libère la liste */
void AL_free(ArrayList * liste);

/* Ajoute un élément en fin */
int AL_add_tail(ArrayList * liste, int value);

/* Ajoute un élément en tête */
int AL_add_head(ArrayList * liste, int value);

/* Ajoute un élément à une position donnée */
int AL_insert(ArrayList * liste, int id, int value);

/* Supprime l'élément en fin */
int AL_pop_tail(ArrayList * liste, int * value);

/* Supprime l'élément en tête */
int AL_pop_head(ArrayList * liste, int * value);

/* Supprime l'élément à une position donnée */
int AL_delete(ArrayList * liste, int id, int * value);

/* Affiche la liste */
void AL_print(FILE * flow, const ArrayList * liste);
```

Exercice 112 (∞ Algorithmes génétiques et problème du voyageur de commerce). L'entreprise ESGI Delivery doit livrer beaucoup de commandes et a besoin de votre aide pour déterminer un tracé qui permettrait ces livraisons sans trop consommer de carburant. Ils souhaitent un programme qui depuis des positions de villes leur permette en quelques secondes d'avoir une solution pertinente pour envoyer un chauffeur effectuer les livraisons.

Le problème proposé est communément connu comme le "problème du voyageur de commerce". Ce problème est connu comme étant un problème algorithmiquement difficile à résoudre (ordre de possibilité de l'ordre de la factorielle du nombre de villes à considérer). Nous vous proposons donc d'implémenter un algorithme génétique pour proposer une solution pertinente : changements aléatoires sur des chemins possibles (mutations), sélection naturelle des meilleures proposition pour duplication et répétition du procédé. Nous avons croisé un chercheur en bio-informatique qui nous a donné l'algorithme suivant :

Algorithm 5: Optimisation génétique du problème du voyageur de commerce

```

input : Entier Taille de la sélection
input : Entier Taille de la génération
input : Chemin chemin : liste de villes (saisie utilisateur)
output: Chemin solution : liste de villes (distance totale optimisée)

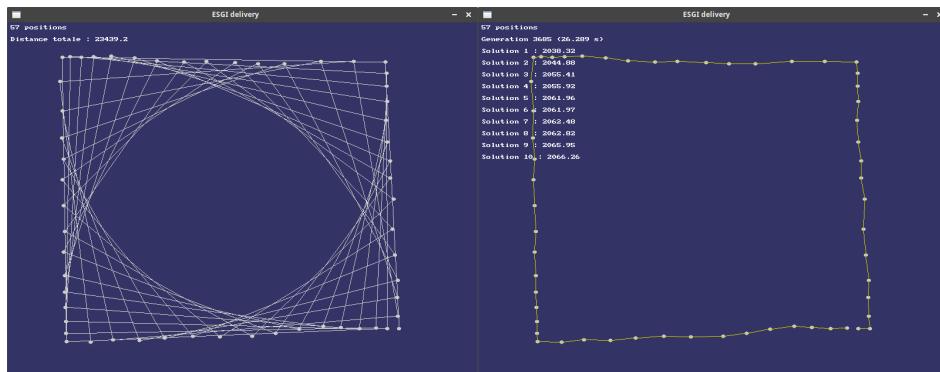
1 Construire Population une liste de Chemin de taille Taille de la génération ;
2 Copier chemin à chaque élément de Population ;
3 Tant que L'utilisateur n'est pas satisfait de solution faire
4   Pour chaque chemin dans Population [Taille de la sélection :] faire
5     | Faire muter chemin ;
6     | Calculer la distance totale de chemin ;
7   Fin
8   Affecter aux Taille de la sélection premiers éléments de Population les
     chemins de plus petite longueur (triés en ordre croissant). ;
9   Pour chaque chemin dans Population [Taille de la sélection :] faire
10    | chemin  $\leftarrow$  Population [indice de chemin modulo Taille de la sélection ] ;
11    Fin
12    solution  $\leftarrow$  Population [0] ;
13 Fin

```

Un stagiaire de l'entreprise avait commencé la mission, mais son stage étant arrivé à échéance, vous devrez compléter son programme et réaliser l'implémentation de l'optimisation du parcours des villes. À noter que nous voudrions tester votre programme sur les fichiers suivants pour nous assurer de son bon fonctionnement puis sur un exemple de ville :

- **star.points** : distance attendue sous 1 500 en 1 seconde (meilleure solution trouvée : 964.232).
- **rectangle.points** : distance attendue sous 3 500 en 10 seconde (meilleure solution trouvée : 2038.32).
- **city.points** : distance attendue sous 9 000 en 30 secondes (meilleure solution trouvée : 7944.17).

L'entreprise aimerait que vous continuiez cette mission et proposiez un programme qui pourrait donner l'affichage suivant (à gauche l'entrée utilisateur, à droite une solution optimisée) :



Voici le code commencé par le stagiaire de l'entreprise :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 600;
const char * titre = "ESGI delivery";
```

```

#define SELECT 10
#define GENERATE 1000

/* TODO : définir un modèle pour sauvegarder la liste utilisateur */

/* TODO : définir la liste utilisateur */

/* Nombre de points saisis par l'utilisateur : */
int taille = 0;

/* TODO : définir un modèle pour les solutions au problème */

/* Nombre de cycles d'optimisation des solutions : */
int generations = 0;
clock_t start;

void Entries_init() {
    /* TODO : initialise la liste de points à optimiser */
}

void Entries_update() {
    /* TODO : optimiser les solutions actuelles */
    ++generations;
}

void addInput(int x, int y) {
    /* TODO : ajouter un point à la liste utilisateur */
    ++taille;
}

void affichage(SDL_Surface * ecran) {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));

    int i;
    double distance = 0.;
    /* TODO : dessiner la liste de points utilisateur et la solution la
       ↪ plus pertinente trouvée */
    /* TODO : calculer la distance totale de la proposition actuelle */
    char buffer[512];
    sprintf(buffer, "%d positions", taille);
    stringRGBA(ecran, 5, 5, buffer, 255, 255, 255, 255);
    if(generations == 0) {
        sprintf(buffer, "Distance totale : %g", distance);
    } else {

```

```
    sprintf(buffer, "Generation %d (%g s)", generations, (double)(clock()
        - start) / CLOCKS_PER_SEC);
}

stringRGBAEcran, 5, 25, buffer, 255, 255, 255, 255);
/* TODO : afficher distance des meilleures solutions */
}

int Points_save(const char * path, ...) {
    /* TODO : sauvegarder la liste de points de l'utilisateur */
    return 0;
}

int Points_load(const char * path, ...) {
    /* TODO : charger la liste de points de l'utilisateur */
    return 0;
}

int main(int argc, char * argv[]) {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
        - SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    /* TODO : charger le fichier de points en argument si précisé : */

    int active = 1;
    int optimise = 0;
    SDL_Event event;

    while(active) {

        affichage(ecran);
        SDL_Flip(ecran);

        while(SDL_PollEvent(&event)) {
```

```
switch(event.type) {
    /* Utilisateur clique sur la croix de la fenêtre : */
    case SDL_QUIT : {
        active = 0;
    } break;

    /* Utilisateur enfonce une touche du clavier : */
    case SDL_KEYDOWN : {
        switch(event.key.keysym.sym) {
            /* Touche Echap : */
            case SDLK_ESCAPE : {
                active = 0;
            } break;
        }
    } break;

    case SDL_KEYUP : {
        switch(event.key.keysym.sym) {
            case SDLK_SPACE : {
                if(! optimise) {
                    Entries_init();
                    start = clock();
                }
                optimise = 1;
            } break;

            case SDLK_s : {
                /* TODO : sauvegarder la liste de points actuelle */
            } break;
        }
    } break;

    case SDL_MOUSEBUTTONDOWN : {
        switch(event.button.button) {
            case SDL_BUTTON_LEFT : {
                if(! optimise) {
                    addInput(event.button.x, event.button.y);
                }
            } break;
        }
    } break;
}

if(optimise) {
```

```
    Entries_update();
    SDL_Delay(1);
} else {
    SDL_Delay(1000 / 60);
}

}

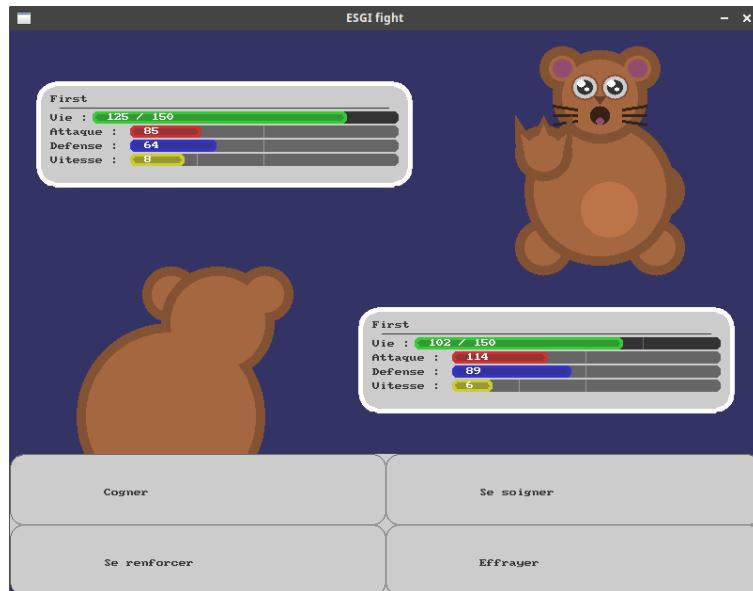
SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

Exercice 113 (∞ Combat de créatures par fichiers). Alice souhaite créer un jeu à la Pokemon. Elle a réussi à afficher quelques formes avec SDL 1.2, mais a du mal à implémenter le chargement de personnages et des capacités. Elle a choisi de proposer le chargement d'un personnage par un fichier :

- Statistiques initiales.
- Dessin du personnage vu de face.
- Dessin du personnage vu de dos.
- Capacités à charger (fichiers avec nom, expressions pour les actions et message à afficher lors de l'activation).

Pour un souci d'optimisation, elle vous indique que ces fichiers ne doivent être chargés qu'avant le combat et que vous devrez proposer une représentation adaptée en mémoire pour appeler l'affichage du personnage et le lancement d'une capacité.

Oscar fait trop le malin et dit que c'est trivial à faire, il lui a envoyé l'image suivante :



Alice a commencé le code suivant, à vous de le compléter pour le rendre fonctionnel :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

/* Paramètres de la fenêtre : */
const int largeur = 800;
const int hauteur = 600;
const char * titre = "ESGI fight";
SDL_Surface * ecran = NULL;

/* Statistiques d'un personnage : */
typedef struct Stats Stats;
struct Stats {
    int vie;
    int atk;
    int def;
    int vit;
};

/* Codes de dessin des formes : */
typedef enum {
    S_NONE      = 0,
    S_LINE      = 1,
    S_CIRCLE    = 2,
    S_POLYGON   = 3
} Shape;

/* Élément d'évaluation d'une action modifiant les stats (voir capacité)
   ↵ : */
typedef struct Action Action;
struct Action {
    /* TODO : champs libres modélisant un noeud */
    Action * left; /* fils gauche d'une expression */
    Action * right; /* fils droit d'une expression */
    Action * next; /* action suivante */
};

int Action_read(Action ** action, const char * start, const char * end) {
    /* TODO : lecture d'une action depuis une sous-chaîne de caractères */
    return 1;
}
```

```

void Action_debug(const Action * action) {
    /* TODO : affichage debug d'une action : vérification de l'expression
       ↵   */
}

float Action_eval(const Action * action, Stats * self_stats, Stats *
    ↵ other_stats) {
    /* TODO : évaluation d'un action */
    return 0.f;
}

/* Sort d'un personnage : */
typedef struct Capacite Capacite;
struct Capacite {
    char nom[64]; /* Nom du sort */
    char message[1024]; /* Message à afficher à l'utilisation */
    Action * action; /* Action à déclencher */
};

int Capacite_load(const char * path, Capacite * cap) {
    /* TODO : lecture d'une capacité */
    return 1;
}

void Capacite_debug(const Capacite * cap) {
    /* TODO : affichage debug d'une capacité */
}

typedef struct Personnage Personnage;
struct Personnage {
    char name[50]; /* nom du personnage */
    Stats start; /* statistiques initiales */
    Stats current; /* statistiques actuelles */
    unsigned char face[1024]; /* codage du dessin de face */
    unsigned char back[1024]; /* codage du dessin de dos */
    Capacite capacites[4]; /* capacités */
    int nb_caps; /* nombre de capacités disponibles */
};

Personnage joueur;
Personnage adversaire;

void Personnage_afficher_stats(const Personnage * perso, int x, int y) {
    /* TODO : afficher les barres de stats d'un personnage depuis l'origine
       ↵   (x, y) */
}

```

```

int sx = x - largeur / 4;
int sy = y - hauteur / 9;
int ex = x + largeur / 4;
int ey = y + hauteur / 13;
roundedBoxRGBA(ecran, sx, sy, ex, ey, 20, 255, 255, 255);
}

void draw_data(unsigned char * data, int cx, int cy, int s) {
    /* TODO : dessiner un modèle vectoriel (lu au chargement d'un
       ↳ personnage) */
    /* (cx, cy) est le centre de l'image et s le diamètre du dessin */
    filledCircleRGBA(ecran, cx, cy, s / 2, 127, 127, 127, 255);
}

int Personnage_load(Personnage * perso, const char * path) {
    /* TODO : charger un personnage */
    perso->nb_caps = 4;
    return 1;
}

void affichage() {
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51, 102));

    Personnage_afficher_stats(&adversaire, 2 * largeur / 7, hauteur / 5);
    Personnage_afficher_stats(&joueur, 5 * largeur / 7, 3 * hauteur / 5);
    draw_data(joueur.back, 3 * largeur / 14, 3 * hauteur / 5, hauteur / 3);
    draw_data(adversaire.face, 11 * largeur / 14, hauteur / 5, hauteur /
        ↳ 3);
}

int afficher_choix(int mx, int my, const Personnage * perso) {
    roundedBoxRGBA(ecran, 0, 3 * hauteur / 4, largeur, hauteur, 15, 204,
        ↳ 204, 204, 255);

    int cap = -1;
    int i, x, y;
    if(mx > 0 && mx < largeur && my > 3 * hauteur / 4 && my < hauteur) {
        x = mx / (largeur / 2);
        y = (my - 3 * hauteur / 4) / (hauteur / 8);
        cap = 2 * y + x;
        if(cap < perso->nb_caps) {
            roundedBoxRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 + y *
                ↳ hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 + (y + 1) *
                ↳ hauteur / 8, 15, 255, 255, 255);
        }
    }
}

```

```

    } else {
        cap = -1;
    }
}
for(i = 0; i < perso->nb_caps; ++i) {
    x = i % 2;
    y = i / 2;
    roundedRectangleRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 + y *
        hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 + (y + 1) *
        hauteur / 8, 15, 102, 102, 102, 255);
    stringRGBA(ecran, (x + 0.25) * largeur / 2, 3 * hauteur / 4 + (y +
        0.5) * hauteur / 8, perso->capacites[i].nom, 51, 51, 51, 255);
}
return cap;
}

void player_turn(int cap) {
    /* TODO : jouer le tour du joueur */
}

void opponent_turn() {
    /* TODO : jouer le tour de l'adversaire */
}

void display_winner() {
    char buffer[1024];
    if(adversaire.current.vie <= 0) {
        roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur / 8, 3 *
            hauteur / 4, 15, 0, 53, 0, 204);
        sprintf(buffer, "Victoire de %s", joueur.name);
    } else {
        roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur / 8, 3 *
            hauteur / 4, 15, 53, 0, 0, 204);
        sprintf(buffer, "Defaite de %s", joueur.name);
    }
    stringRGBA(ecran, 2 * largeur / 8, hauteur / 2, buffer, 255, 255, 255,
        255);
}

int finished() {
    /* TODO : condition de fin du combat */
    return 0;
}

int main() {

```

```
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
→     SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    int active = 1;
    SDL_Event event;

    if(! Personnage_load(&joueur, "first.perso")
    || ! Personnage_load(&adversaire, "first.perso")) {
        goto end;
    }

    int last_mouse_x = 0;
    int last_mouse_y = 0;
    int cap = -1;
    int select_cap = 0;

    while(active) {

        affichage();
        cap = afficher_choix(last_mouse_x, last_mouse_y, &joueur);
        SDL_Flip(ecran);

        while(SDL_PollEvent(&event)) {

            switch(event.type) {
                /* Utilisateur clique sur la croix de la fenêtre : */
                case SDL_QUIT : {
                    active = 0;
                } break;

                /* Utilisateur enfonce une touche du clavier : */
                case SDL_KEYDOWN : {
                    switch(event.key.keysym.sym) {
                        /* Touche Echap : */

```

```
        case SDLK_ESCAPE : {
            active = 0;
        } break;

        default : break;
    }
} break;

case SDL_MOUSEMOTION : {
    last_mouse_x = event.motion.x;
    last_mouse_y = event.motion.y;
} break;

case SDL_MOUSEBUTTONDOWN : {
    select_cap = 1;
} break;

default : break;
}
}

if(select_cap && cap >= 0 && cap < joueur.nb_caps) {
    if(joueur.current.vit >= adversaire.current.vit) {
        player_turn(cap);
        opponent_turn();
    } else {
        opponent_turn();
        player_turn(cap);
    }
    active = ! finished();

    select_cap = 0;

    if(! active) {
        display_winner();
        SDL_Flip(ecran);
        SDL_Delay(2000);
    }
}

SDL_Delay(1000 / 60);

end:
```

```
SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

Voici un exemple de fichier personnage :

```
name : Anonymous
vie : 150
atk : 80
def : 50
vit : 10
face : {
    circle -30, 55, 15 rgba 131, 82, 51, 255
    circle -30, 55, 10 rgba 165, 103, 64, 255
    circle 30, 55, 15 rgba 131, 82, 51, 255
    circle 30, 55, 10 rgba 165, 103, 64, 255
    circle 30, 25, 15 rgba 131, 82, 51, 255
    circle 30, 25, 10 rgba 165, 103, 64, 255
    circle 0, 25, 40 rgba 131, 82, 51, 255
    circle 0, 25, 35 rgba 165, 103, 64, 255
    circle 5, 35, 15 rgba 187, 117, 72, 255
    circle 20, -40, 12 rgba 131, 82, 51, 255
    circle -20, -40, 12 rgba 131, 82, 51, 255
    circle 0, -25, 25 rgba 131, 82, 51, 255
    circle 20, -40, 7 rgba 165, 103, 64, 255
    circle -20, -40, 7 rgba 165, 103, 64, 255
    circle 20, -40, 5 rgba 146, 77, 110, 255
    circle -20, -40, 5 rgba 146, 77, 110, 255
    circle 0, -25, 20 rgba 165, 103, 64, 255
    circle 8, -30, 7 rgba 131, 82, 51, 255
    circle -8, -30, 7 rgba 131, 82, 51, 255
    circle 8, -30, 6 rgba 204, 204, 204, 255
    circle -8, -30, 6 rgba 204, 204, 204, 255
    circle 8, -30, 4 rgba 51, 51, 51, 255
    circle -8, -30, 4 rgba 51, 51, 51, 255
    circle 6, -32, 2 rgba 255, 255, 255, 255
    circle 9, -29, 1 rgba 255, 255, 255, 255
    circle -10, -32, 2 rgba 255, 255, 255, 255
    circle -7, -29, 1 rgba 255, 255, 255, 255
    circle 0, -15, 5 rgba 54, 34, 21, 255
    circle 0, -12, 2 rgba 146, 77, 110, 255
    circle -30, 5, 15 rgba 131, 82, 51, 255
    line 12, -18, 25, -20, 1 rgba 54, 34, 21, 255
}
```

```

line 12, -15, 25, -15, 1 rgba 54, 34, 21, 255
line 12, -12, 25, -10, 1 rgba 54, 34, 21, 255
line -12, -18, -25, -20, 1 rgba 54, 34, 21, 255
line -12, -15, -25, -15, 1 rgba 54, 34, 21, 255
line -12, -12, -25, -10, 1 rgba 54, 34, 21, 255
polygon 3 3, -25, -3, -25, 0, -21 rgba 104, 65, 40, 255
polygon 3 -35, -5, -25, -5, -30, -15 rgba 131, 82, 51, 255
polygon 3 -25, -0, -15, 0, -15, -15 rgba 131, 82, 51, 255
polygon 3 -35, -0, -45, 0, -45, -15 rgba 131, 82, 51, 255
polygon 3 3, -25, -3, -25, 0, -21 rgba 104, 65, 40, 255
circle -30, 5, 10 rgba 165, 103, 64, 255
polygon 3 -32, -5, -28, -5, -30, -10 rgba 165, 103, 64, 255
polygon 3 -30, 5, -20, 5, -18, -5 rgba 165, 103, 64, 255
polygon 3 -30, 5, -40, 5, -42, -5 rgba 165, 103, 64, 255
}
back : {
circle 0, 25, 50 rgba 131, 82, 51, 255
circle 0, -40, 15 rgba 131, 82, 51, 255
circle 0, -40, 10 rgba 165, 103, 64, 255
circle 25, -25, 30 rgba 131, 82, 51, 255
circle 50, -40, 15 rgba 131, 82, 51, 255
circle 25, -25, 25 rgba 165, 103, 64, 255
circle 50, -40, 10 rgba 165, 103, 64, 255
circle 0, 25, 45 rgba 165, 103, 64, 255
}
capacities : {
load coup.cap
load heal.cap
load raise.cap
load fear.cap
}

```

Voici un exemple de fichier capacité (`coup.cap`) :

```

nom Cogner
action $other.vie -= ($self.atk * 12.5 + 50) / $other.def + 1
message $self cogne $other .

```

Attendus :

- Lecture d'une capacité et modélisation d'une expression en mémoire.
- Évaluation et application d'une capacité.
- Modélisation d'une image vectorielle en mémoire et dessin d'un personnage.

- Lecture d'un personnage et dessin des statistiques d'un personnage.
- Jouer et terminer un combat.

12 Programmation modulaire

12.1 Retour sur la compilation

12.1.1 Externaliser une fonction

Jusqu'ici, nous avons essentiellement travaillé sur des travaux pratiques pour lesquels écrire le code dans une seul fichier semblait faire l'affaire. Cependant pour un projet ou quand le code commence à devenir plus conséquent et plus complexe, se limiter à un fichier peut vite devenir douloureux.

Prenons l'exemple du code suivant :

```
#include <stdio.h>
#include <stdlib.h>

void maFonction() {
    printf("Ma Fonction\n");
}

int main() {
    maFonction();
    exit(EXIT_SUCCESS);
}
```

Ce code définit une fonction puis l'appel. Imaginons que nous ne voudrions pas définir cette fonction dans ce fichier, mais dans un autre. Ceci est possible :

- Nous avons besoin de la déclaration de la fonction pour l'appeler dans notre code.
- Sa définition peut être sauvegardée dans un autre fichier.

main.c

```
#include <stdlib.h>

extern void maFonction();

int main() {
    maFonction();
    exit(EXIT_SUCCESS);
}
```

mes_fonctions.c

```
#include <stdio.h>

void maFonction() {
    printf("Ma Fonction\n");
}
```

Pour compiler une telle configuration, il faut que les deux fichiers soient compilés et liés vers le même exécutable. Ceci se fait en ajoutant `mes\fonctions.c` dans la ligne de compilation :

```
gcc -o executable main.c mes_fonctions.c
```

`maFonction` a maintenant été externalisée dans un autre fichier. Le mot-clé `extern` est facultatif pour une fonction. Cependant, il est nécessaire si l'on souhaite utiliser une variable instanciée dans un autre fichier source :

main.c

```
#include <stdlib.h>

extern int variable;

extern void maFonction(int);

int main() {
    maFonction(variable);
    exit(EXIT_SUCCESS);
}
```

mes_fonctions.c

```
#include <stdio.h>

int variable = 42;

void maFonction(int v) {
    printf("Ma Fonction %d\n", v);
}
```

12.1.2 Compilation séparée

Revenons sur la compilation. Rappelez vous, nous avions parlé de compilateur et d'éditeur des liens :

- **La compilation** c'est la transformation du code source en langage C en langage utilisable par la machine. Ceci produira des fichiers "Objets" avec pour extension .o.
- **L'édition des liens** c'est l'assemblage de ces fichiers .o en une application (exécutable) et le raccordement des définitions de chaque élément à sa déclaration et son appel.

S'il reste une déclaration non résolue (absence du fichier contenant la définition d'une fonction par exemple), le compilateur vous l'indiquera à l'édition des liens :

```
# gcc -o executable main.c
/tmp/cc5vxQgu.o : Dans la fonction « main » :
main.c:(.text+0xa) : référence indéfinie vers « maFonction »
collect2: error: ld returned 1 exit status
```

Nous avons vu ici une version simplifiée de la commande de compilation : tout est compilé puis linké. Cependant ceci veut dire que pour la modification d'un fichier, l'ensemble est recompilé. Il est donc possible de compiler les fichiers uns à uns avec l'option `-c`. Celle-ci fabrique un fichier `.o` associé à chaque fichier `.c` fourni. Ensuite on procède à l'édition des liens en fournissant les fichiers `.o` et bibliothèques utilisées.

```
# gcc -c mes_fonctions.c
# gcc -c main.c
# gcc -o executable main.o mes_fonctions.o
```

Si vous souhaitez visualiser la table des symboles d'un fichier `.o`, vous pouvez utiliser la commande `readelf` :

```
# readelf -s main.o

La table de symboles « .symtab » contient 13 entrées :
  Num:    Valeur          Tail Type   Lien   Vis      Ndx Nom
    0: 00000000000000000000    0 NOTYPE LOCAL  DEFAULT  UND
    1: 00000000000000000000    0 FILE    LOCAL  DEFAULT  ABS main.c
...
    8: 00000000000000000000   27 FUNC    GLOBAL DEFAULT  1 main
    9: 00000000000000000000    0 NOTYPE GLOBAL DEFAULT  UND
      → variable
   10: 00000000000000000000    0 NOTYPE GLOBAL DEFAULT  UND
      → _GLOBAL_OFFSET_TABLE_
   11: 00000000000000000000    0 NOTYPE GLOBAL DEFAULT  UND
      → maFonction
   12: 00000000000000000000    0 NOTYPE GLOBAL DEFAULT  UND exit
```

Nous utilisons aussi `printf`, une fonction définie ailleurs, sans l'avoir déclarée, non ? Vos fichiers ne sont en réalité pas si vides et vous l'avez fait avec votre

`#include <stdio.h>`. C'est une directive préprocesseur, voyons comment ceci fonctionne.

12.2 Directives préprocesseur

Un programme qui affiche "Hello ESGI!", ça tient en 10 lignes, regardez :

```
#include <stdio.h>

int main() {
    printf("Hello ESGI !\n");
    return 0;
}
```

Essayez maintenant la commande suivante :

```
gcc -o main.i -E -P main.c
```

Ceci ressemble plutôt à ça :

```
typedef long unsigned int size_t;
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
... /* environ 200 lignes */

int main() {
    printf("Hello ESGI !\n");
    return 0;
}
```

Cette commande permet de résoudre les directives préprocesseurs. En effet, avant d'être compilé, un code source en langage C est pré-traité pour qu'il ne reste réellement que des instructions en langage C. Ces directives préprocesseur permettent de gagner en lisibilité ou maintenabilité lorsque nous avons besoin que des informations soient inscrites en dur dans du code C. Dans notre cas, ceci nous permet de déclarer toutes les fonctionnalités que nous utiliserons de la bibliothèques `stdio.h` depuis son entête de déclaration. Notez que simple déclarer `printf` fonctionne sans erreur ni warning :

```
extern int printf(const char *, ...);

int main() {
    printf("Hello ESGI !\n");
    return 0;
}
```

12.2.1 include

La directive préprocesseur `include` permet de recopier le contenu d'un fichier à l'emplacement du fichier C courant où elle est appelée. C'est un outil puissant, mais à utiliser avec parcimonie. En général, nous les utiliserons pour inclure des fichiers d'entête dans notre code :

- `#include <entete.h>` : permet l'inclusion d'une bibliothèque présente ou installée sur la machine.
- `#include "entete.h"` : est réservé à l'inclusion relative de vos propres fichiers d'entête.

Ceci permet par exemple de définir sa propre bibliothèque de fonctionnalités :

```
main.c

#include <stdlib.h>
#include "mes_fonctions.h"

int main() {
    maFonction(variable);
    exit(EXIT_SUCCESS);
}
```

```
mes_fonctions.h
```

```
extern int variable;  
  
extern void maFonction(int);
```

```
mes_fonctions.c
```

```
#include <stdio.h>  
  
int variable = 42;  
  
void maFonction(int v) {  
    printf("Ma Fonction %d\n", v);  
}
```

Nous verrons dans la suite comment protéger cette bibliothèque contre des problèmes de doublons dans l'inclusion et en faire ainsi un module.

12.2.2 define

Nous avions déjà croisé une utilisation de la directive préprocesseur `define` pour la création de constantes. Cette directive fera que pour un nom donné, celui-ci sera remplacé par le code associé.

Expressions constantes

Ceci permet par exemple de déclarer des expression constantes :

```
#include <stdio.h>  
#include <stdlib.h>  
#define TAILLE 10  
  
int main() {  
    int i;  
    for(i = 0; i < TAILLE; ++i) {
```

```

    printf("%d\n", i);
}
exit(EXIT_SUCCESS);
}

```

En général, on les introduit plutôt en début de fichier, mais ces directives peuvent être placées où souhaité dans le code. Leur définition peut être annulée avec la directive préprocesseur `undef` et elles peuvent être redéfinies :

```

#include <stdio.h>
#include <stdlib.h>

int main() {
#define NOMBRE 42
    printf("define NOMBRE %d\n", NOMBRE);
#undef NOMBRE
#define NOMBRE 13.37
    printf("define NOMBRE %g\n", NOMBRE);
#undef NOMBRE
#define NOMBRE "1234"
    printf("define NOMBRE %s ?\n", NOMBRE);
    exit(EXIT_SUCCESS);
}

```

La directive `define` peut être utilisée pour remplacer un nom de fonction ou même du code :

```

#include <stdio.h>
#include <stdlib.h>

#define print puts

int main() {
    print("Hello ESGI");
    /* Python de langage C ! */
    exit(EXIT_SUCCESS);
}

```

```
}
```

Pour revenir à la ligne dans le contenu associé à un `#define`, il faut utiliser un anti-slash :

```
#include <stdio.h>
#include <stdlib.h>

#define INSTRUCTIONS printf("Hello ESGI\n"); \
                           exit(EXIT_SUCCESS);

int main() {
    INSTRUCTIONS
}
```

Macros

Il est possible de définir des "Macros" à l'aide du `#define`. Ces macros vont écrire en dur du code en fonction de paramètres données. Ceci peut être utilisé par exemple pour remplacer des fonctions courtes garder une générnicité sur les arguments fournis.

```
#include <stdio.h>
#include <stdlib.h>

#define min(a, b) (a < b) ? a : b

int main() {
    int a = 42, b = 1337;
    printf("min(%d, %d) = %d\n", a, b, min(a, b));
    float c = 13.37, d = 42.;
    printf("min(%g, %g) = %g\n", c, d, min(c, d));
    exit(EXIT_SUCCESS);
}
```

Cependant, ceci reste à utiliser avec parcimonie. En effet, la macro va réécrire à l'identique la partie de code fournie en argument, là où une fonction travaillerait

avec la valeur de retour fournie. Il est donc conseillé de mettre les arguments entre parenthèses dans le code de la macro et d'être rigoureux avec les arguments dont la duplication de l'expression pourrait être problématique :

```
int fonction_min(int a, int b) {
    return (a < b) ? a : b;
}

#define macro_min(a, b) ((a) < (b)) ? (a) : (b)

int main() {
    int v, a, b;
    v = fonction_min(a = getchar() - '0', b = getchar() - '0');
    printf("fonction_min(%d, %d) = %d\n", a, b, v);
    v = macro_min(a = getchar() - '0', b = getchar() - '0');
    printf("macro_min(%d, %d) = %d\n", a, b, v);
    exit(EXIT_SUCCESS);
}
```

Ce code produit en réalité le code suivant :

```
int fonction_min(int a, int b) {
    return (a < b) ? a : b;
}

int main() {
    int v, a, b;
    v = fonction_min(a = getchar() - '0', b = getchar() - '0');
    printf("fonction_min(%d, %d) = %d\n", a, b, v);
    v = ((a = getchar() - '0') < (b = getchar() - '0')) ? (a =
        ↳ getchar() - '0') : (b = getchar() - '0');
    printf("macro_min(%d, %d) = %d\n", a, b, v);
    exit(0);
}
```

D'où une exécution potentiellement erronée :

```
24246
fonction_min(4, 2) = 2
macro_min(6, 4) = 6
```

Il est possible de transformer une expression fournie à une macro en une chaîne de caractère en précédant le nom de l'expression par un dièse # dans le code associé à la macro :

```
#define FAIRE_CALCUL(exp) printf("%s = %d\n", #exp, exp)

int main() {
    int valeur = 4;
    FAIRE_CALCUL(valeur * valeur + 2 * valeur + 1);
    exit(EXIT_SUCCESS);
}
```

Ce qui produit l'exécution suivante :

```
valeur * valeur + 2 * valeur + 1 = 25
```

Pour concaténer l'expression fournie à une macro avec du code : par exemple dans le nom d'une fonction. Il est possible d'appeler d'appeler l'opérateur ## dans le code associé à la macro :

```
#define macro_abs(a) (((a) < 0) ? -(a) : (a))
#define macro_carre(a) ((a) * (a))

#define call(f, x) printf("%s(%d) = %d\n", #f, x, macro_##f(x))

int main() {
    call(abs, -5);
    call(carre, -5);
    exit(EXIT_SUCCESS);
}
```

Ce qui produit l'exécution suivante :

```
abs(-5) = 5
carre(-5) = 25
```

12.2.3 conditionnement

Il est possible de conditionner du code sous condition. Par exemple, il peut être utile d'avoir un mode verbeux pour travailler sur votre code et un mode moins verbeux pour le fournir à un utilisateur.

Par exemple dans le code suivant, on peut adapter le code en fonction du fait que certaines `#define` existent ou non :

```
#define DEBUG

int main() {
#if defined VERBOSE
    fprintf(stderr, "Entrée dans le main\n");
#endif
#if defined VERBOSE
    printf("42, la réponse à la vie !\n");
#elif defined DEBUG
    printf("42, vous savez pourquoi.\n");
#else
    printf("42 !\n");
#endif
#if defined VERBOSE
    fprintf(stderr, "Sortie du main\n");
#endif
    exit(EXIT_SUCCESS);
}
```

Ceci produit le code suivant :

```
int main() {
    printf("42, vous savez pourquoi.\n");
```

```
    exit(0);
}
```

Il est possible de réaliser un `#define` depuis la commande de compilation avec l'option `-D` :

```
gcc -o main.i -E -P main.c -DVERBOSE
```

Ce qui produit le code suivant :

```
int main() {
    fprintf(stderr, "Entrée dans le main\n");
    printf("42, la réponse à la vie !\n");
    fprintf(stderr, "Sortie du main\n");
    exit(0);
}
```

Il est possible de raccourcir l'écriture des conditions vérifiant la définition d'un `#define` :

```
#if defined NOM
#endif NOM

#if !defined NOM
#endif NOM
```

12.2.4 Module

Nous avons vu précédemment qu'il peut être intéressant de répartir ses fonctionnalités dans d'autres fichiers. Plus précisément nous découperons notre code en **modules**. Un module est un ensemble de fonctionnalités documentées pour lequel on fournit un fichier d'entête `.h` à l'utilisateur (un autre programmeur ou vous-même) et pour lequel vous avez implémenté les fonctionnalités associées dans un fichier `.c` du même nom.

Exemple d'organisation d'un module :

module.h

```
#ifndef DEF_HEADER_MODULE
#define DEF_HEADER_MODULE
/* Protection du module */

/**
 Documentation du module et auteurs
 */

/* Macros publiques */
#define abs(x) (((x) < 0) ? -(x) : (x))

/* Types publiques du module */
typedef struct Point Point;
struct Point {
    float x;
    float y;
};

/* Variables publiques du module */
extern Point origine;

/* Fonctionnalités publiques du module */

/* Affiche un point dans la sortie standard */
extern void Point_afficher(const Point * point);

#endif
```

module.c

```
#include "module.h"
/* Inclusion des déclarations du module */

#include <stdio.h>
/* Autres inclusions */

/* Définition des variables globales relatives au module
→ */
Point origine = {0, 0};

/* Fonctionnalité privée au module par le mot-clé static
→ */
static void Point_print(FILE * flow, const Point * point)
→ {
    if(! point) {
        fprintf(flow, "(nil)");
    }
    fprintf(flow, "(%g, %g)", point->x, point->y);
}

/* Définition des fonctionnalités annoncées par l'entête
→ */
void Point_afficher(const Point * point) {
    Point_print(stdout, point);
}
```

12.3 Makefiles

12.3.1 Concept

Avec l'augmentation du nom de modules, l'ajout de bibliothèques et autres la compilation de votre projet va se complexifier. Pour ceci, il peut être intéressant d'automatiser sa compilation. À noter qu'un avantage offert par la programmation modulaire est que l'on peut travailler sur chaque module indépendamment des interdépendances entre les modules. Et donc avoir une compilation potentiellement plus rapide que recompiler l'intégralité du projet.

Le moyen proposé pour compiler un projet en langage C est un **Makefile**. Un Makefile est un fichier qui donne le schéma à suivre pour compiler le code. Celui-ci demande d'être créé à l'emplacement d'où vous souhaitez que l'utilisateur compile votre projet. Il n'aura ensuite besoin que d'utiliser la commande `make`.

Makefile

```
executable :  
    gcc -o executable *.c
```

make

12.3.2 Possibilités

Un Makefile s'organise comme une liste de cibles, dépendances associées et commandes à exécuter. Chaque commande est précédée d'une **tabulation**.

```
cible: dépendances  
      commande  
      ...  
      commande
```

Les dépendances vont permettre de relancer les commandes pour construire la cible si elles ont été modifiées.

La proposition de compilation est la suivante :

- Compilation séparée des fichiers .c en fichiers .o pour chaque module.
 - Fichier source .c du module et fichiers .h inclus dans le module en dépendances.
 - Linkage de tous les fichiers .o pour création de l'exécutable.
 - Une commande pour nettoyer le répertoire de la compilation.
- Ceci pourrait donner le Makefile suivant :

```
executable : main.o module.o
    gcc -o executable main.o module.o

main.o : main.c module.h
    gcc -c main.c

module.o : module.c module.h
    gcc -c module.c

clean :
    rm -rf *.o
```

La commande de nettoyage du dossier peut s'appeler de la manière suivante :

```
make clean
```

Cependant ce Makefile peut être assez redondant, il est possible d'introduire les symboles suivants :

Symbol	Correspondance
\$@	Cible
\$^	Toutes les dépendances
\$<	Première dépendance

Ceci permet la transformation du Makefile en le suivant :

```
executable : main.o module.o
    gcc -o $@ $^
```

```
main.o : main.c module.h
    gcc -c $<

module.o : module.c module.h
    gcc -c $<
```

Pour aller plus loin, il est aussi possible d'utiliser une règle générique en utilisant un %. Celui-ci sera automatiquement remplacé de manière à construire les cibles manquantes correspondant au schéma donné. Cependant, il sera nécessaire d'ajouter les dépendances pour qu'elles soient prises en compte :

```
executable : main.o module.o
    gcc -o $@ $^

main.o : main.c module.h
module.o : module.c module.h

%.o : %.c
    gcc -c $<
```

Il est aussi possible de définir des variables pour votre Makefile. Leur définition prend la syntaxe VARIABLE=VALEUR et leur appel \$(VARIABLE). Elles sont utiles pour écrire à un endroit :

- le compilateur : CC=gcc (ou par exemple CC=clang).
- les drapeaux de compilation et optimisations dans CFLAGS (les optimisations peuvent être -O1, -O2 et éventuellement -O3).
- les bibliothèques dans CLIBS.
- le nom de l'exécutable.

```
CC= gcc
CFLAGS= -O2 -Wall -Wextra -Werror -ansi
CLIBS= -lm
EXE= executable

$(EXE) : main.o module.o
    $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
```

```
main.o : main.c module.h
module.o : module.c module.h

%.o : %.c
    $(CC) $(CFLAGS) -c $<
```

En langage C, on structurera un projet en séparant les différents fichiers dans des répertoires :

- **include** contient vos fichiers d'entêtes .h.
- **src** contient vos fichiers sources .c.
- **obj** permet de sauvegarder les fichiers compilés .o.

L'exécutable sera ainsi généré à la racine du projet avec les fichiers pertinents : README.md, Makefile et autres. Ceci donne le Makefile suivant :

```
CC= gcc
CFLAGS= -O2 -Wall -Wextra -Werror -ansi
CLIBS= -lm
EXE= executable
OBJ= obj/
SRC= src/
INCL= include/

$(EXE) : $(OBJ)main.o $(OBJ)module.o
        $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)

$(OBJ)main.o : $(SRC)main.c $(INCL)module.h
$(OBJ)module.o : $(SRC)module.c $(INCL)module.h

$(OBJ)%.o : $(SRC)%.c
        $(CC) $(CFLAGS) -o $@ -c $<

clean :
        rm -rf $(OBJ)*.o
        rm -rf $(EXE)
```

Dans le cas où un temps de compilation potentiellement un peu plus important ne vous serait pas dérangeant, il est possible de gagner du temps dans la construction du Makefile en utilisant la fonction `wildcard`. Ceci permet d'obtenir une liste d'éléments suivant un schéma donné. En l'occurrence, dans notre cas obtenir les fichiers sources et entêtes. Et pour la génération de la liste des fichiers objets la fonction `patsubst` qui fera la substitution des `.c` en `.o` dans la listes des sources obtenue. Ceci fait que le Makefile suivant ne vous demandera pas de rajouter une dépendance à chaque ajout de module :

```

CC= gcc
CFLAGS= -O2 -Wall -Wextra -Werror -ansi
CLIBS= -lm
EXE= executable
OBJ= obj/
SRC= src/
INCL= include/
FILEC:= $(wildcard $(SRC)*.c)
FILEH:= $(wildcard $(INCL)*.h)
FILEO:= $(patsubst $(SRC)%.*,$(OBJ)%.o,$(FILEC))

$(EXE) : $(FILEO)
    $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)

$(OBJ)main.o : $(SRC)main.c $(FILEH)
    $(CC) $(CFLAGS) -o $@ -c $<

$(OBJ)%.* : $(SRC)%.* $(INCL)%.*.h
    $(CC) $(CFLAGS) -o $@ -c $<

clean :
    rm -rf $(OBJ)*.*
    rm -rf $(EXE)

```

12.4 Résumé

Les directives préprocesseur sont des instructions traitées avant la compilation. Celles-ci permettent notamment :

```
#include <lib.h> /* copie l'entête d'une bibliothèque lib */
#include "module.h" /* copie l'entête de mon fichier module.h */
#define NOM EXPRESSION /* remplacera NOM dans le code par
→ EXPRESSION */
#undef NOM /* supprime la définition de NOM dans le code qui suit
→ */
```

La directive `define` peut prendre des paramètres dont les expression données lors de l'appel remplacera leur occurrence dans l'expression donnée dans la définition.

```
#define MACRO(exp) exp \ /* remplace exp par le code donné en
→ argument et '\' permet de continuer la macro à la ligne */
#exp \ /* fabrique une chaîne de caractères de exp */
##exp## \ /* concatène exp avec le code adjacent dans la macro
→ */
```

Il est possible d'utiliser des structures conditionnelles avec les directives préprocesseur :

```
#if CONDITION1
/* instructions1 */
#elif CONDITION2
/* instructions2 */
#else
/* instructions3 */
#endif
```

Le mot clé `defined` permet d'indiquer si une `define` a nommé ce nom. Celui-ci peut être abrégé dans les structures conditionnelles :

```
#if defined NOM
#endif
```

```
#if ! defined NOM  
#ifndef NOM
```

On peut diviser son code en modules : un fichier .c (implémentation des fonctionnalités) et .h (déclarations à inclure pour l'utiliser dans un autre fichier source) du même nom. Pour compiler un code découpé en modules, on utilise un **Makefile** :

```
cible: dépendances  
        commandes
```

```
$@ # Cible  
$^ # Toutes les dépendances  
$< # Première dépendance  
VARIABLE= valeur # création d'une variable  
$(VARIABLE) # appel d'une variable
```

12.5 Entraînement

Exercice 114 (★★ Mise en place d'un Makefile).

Découper le code suivant en modules et écrire un Makefile permettant sa compilation modulaire :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct Point Point;
struct Point {
    double x;
    double y;
};

Point Point_creer(double x, double y) {
    Point p = {x, y};
    return p;
}

void Point_afficher(const Point * p) {
    printf("(%.g, %.g)", p->x, p->y);
}

double Point_distance(const Point * first, const Point * second) {
    return sqrt(pow(second->x - first->x, 2) + pow(second->y -
        first->y, 2));
}

typedef struct Triangle Triangle;
struct Triangle {
    Point first;
    Point second;
    Point third;
};

Triangle Triangle_creer(Point first, Point second, Point third) {
    Triangle triangle = {first, second, third};
    return triangle;
```

```
}

void Triangle_afficher(const Triangle * triangle) {
    printf("{Triangle : ");
    Point_afficher(&(triangle->first));
    printf(", ");
    Point_afficher(&(triangle->second));
    printf(", ");
    Point_afficher(&(triangle->third));
    printf("}");
}

double Triangle_perimetre(const Triangle * triangle) {
    return
        Point_distance(&(triangle->first), &(triangle->second))
        + Point_distance(&(triangle->second), &(triangle->third))
        + Point_distance(&(triangle->third), &(triangle->first));
}

int main() {
    Triangle triangle = Triangle_creer(
        Point_creer(0, 0),
        Point_creer(4, 0),
        Point_creer(0, 3)
    );
    Triangle_afficher(&triangle);
    printf("\nPerimetre : %g\n", Triangle_perimetre(&triangle));
    exit(EXIT_SUCCESS);
}
```

Exercice 115 (★★★ Profilage d'un code).

Écrire un fichier d'entête `profile.h` dont les macros permettent un profilage d'un code qui les utiliseraient :

- `DO_PROFILE` active l'aspect verbeux du profilage (inactif sinon).
- `START_FUNCTION` prend le type de retour, le nom et les arguments de la fonction.
- `END_FUNCTION` prend le retour de la fonction.

```
#include <stdio.h>
#include <stdlib.h>

#define DO_PROFILE
#include "profile.h"

START_FUNCTION(int, f, int x)
END_FUNCTION(x * x)

START_FUNCTION(int, main)
    int a = 42;
    a = f(a);
    printf("a = %d\n", a);
    exit(EXIT_SUCCESS);
END_FUNCTION(0)
```

Le code ci-dessus se comporte comme le code ci-dessous :

```
#include <stdio.h>
#include <stdlib.h>

int f(int x) {
    return x * x;
}

int main() {
    int a = 42;
    a = f(a);
    printf("a = %d\n", a);
    exit(EXIT_SUCCESS);
    return 0;
}
```

Si la définition préprocesseur `DO_PROFILE` n'est pas définie, le code donne la sortie suivante :

```
a = 1764
```

Si `DO_PROFILE` est définie, le code se montre plus verbeux et affiche les informations additionnelles suivantes dans la sortie standard d'erreur :

```
# Definition of int function main() in file "main.c" at line 10
< Starting function main :
# Definition of int function f(int x) in file "main.c" at line 7
< Starting function f :
> Ending function f with return x * x
a = 1764
> Exit in function main at line 14 with value EXIT_SUCCESS
```

Renseignez vous sur quelques directives préprocesseur standard :

- `--FUNCTION--`
- `--FILE--`
- `--LINE--`
- `--VA_ARGS--`

Exercice 116 (★ Listes chaînées et ajouts).

Compléter le code suivant pour de sorte de :

- Allouer un maillon individuel via `Node_alloc`.
- Libérer toute la liste chaînée via `Node_free`.
- Ajouter un maillon à un liste chaînée via `Node_ajouter`.
- Afficher une liste chaînée via `Node_afficher`.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node Node;
struct Node {
    int value;
    Node * next;
};

Node * Node_alloc(int value) {
    /* TODO : allouer un maillon d'un liste chaînée */
}

void Node_free(Node ** liste) {
    /* TODO : libérer une liste chaînée */
}

int Node_ajouter(Node ** liste, int value) {
    /* TODO : ajouter un élément à une liste chaînée */
}

void Node_afficher(const Node * liste) {
    printf("[");
    /* TODO : afficher les éléments de la liste chaînée */
    printf("]\n");
}

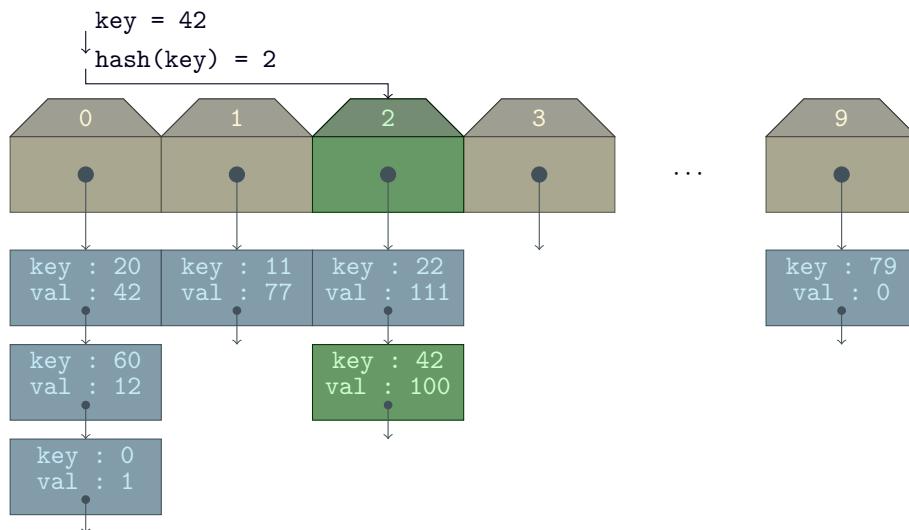
int main() {
    Node * liste = NULL;
    int valeurs[] = {1, 2, 3, 4, -1};
    int i;
    for(i = 0; valeurs[i] >= 0; ++i) {
```

```
if(! Node_ajouter(&liste, valeurs[i])) {
    Node_free(&liste);
    fprintf(stderr, "Erreur ajout liste : arrêt.\n");
    exit(EXIT_FAILURE);
}
Node_afficher(liste);
Node_free(&liste);
exit(EXIT_SUCCESS);
}
```

Exercice 117 (★★★ Implémentation table de hachage).

Nous vous proposons d'implémenter une table de hachage. Une table de hachage est une structure de données qui permet d'associer une **clé** à une **valeur**. Nous proposons ici de rester sur des clés entières et des valeurs entières.

Une table de hachage va réserver un tableau d'éléments d'une capacité donnée. Chaque élément sera accessible via une clé sur laquelle on applique une fonction de hachage. Pour les entiers, nous pouvons garder sa valeur et procéder modulo la capacité. Une fois l'emplacement donné par le hachage de la clé, nous allons à l'indice correspondant dans le tableau d'éléments. Ces éléments peuvent être au choix des tableaux ou des listes chaînées contenant les deux informations que sont la clé et la valeur associée.



La table ci-dessus correspond aux associations suivantes :
 $\{0 \mapsto 1, 11 \mapsto 77, 20 \mapsto 42, 22 \mapsto 111, 42 \mapsto 100, 60 \mapsto 12, 77 \mapsto 0\}$.

Nous aimerais comparer les performances de deux structures de données pour le comptage d'éléments et la recherche de ceux-ci. Un fichier `main.c` vous est donné pour effectuer la comparaison entre une table de hachage implémentée dans un module `hashmap` et un tableau dans un module `arraylist`. Les fichiers d'entête des modules vous sont donnés pour implémentation (à ne pas modifier) :

```

/* fichier hashmap.h */
#ifndef DEF_HEADER_HASHMAP
#define DEF_HEADER_HASHMAP

#include <stdio.h>

/* Table de hachage */
typedef struct HashMap HashMap;
struct HashMap;

/* création d'une table de hachage */
HashMap * HashMap_creer(int capacite);

/* libération d'une table de hachage */
void HashMap_free(HashMap ** hashmap);

/* ajout de 1 à la valeur associée à key, affectation à 1 si non
   → trouvée */
int HashMap_ajouter(HashMap * hashmap, int key);

/* affiche la table de hachage dans un flux flow */
void HashMap_afficher(FILE * flow, const HashMap * hashmap);

/* renvoie le nombre d'ajouts de la clé key (valeur associée) */
int HashMap_compter(const HashMap * hashmap, int key);

#endif

```

```

/* fichier arraylist.h */
#ifndef DEF_HEADER_LISTE
#define DEF_HEADER_LISTE

#include <stdio.h>

/* Liste sous forme d'un tableau de valeurs */
typedef struct ArrayList ArrayList;
struct ArrayList;

/* création d'une liste */
ArrayList * ArrayList_creer(int capacite);

```

```
/* libération d'une liste */
void ArrayList_free(ArrayList ** arraylist);

/* ajout d'un élément dans la liste */
int ArrayList_ajouter(ArrayList * liste, int valeur);

/* affiche la liste dans un flux flow */
void ArrayList_afficher(FILE * flow, const ArrayList * liste);

/* compte le nombre d'occurrences d'une valeur dans la liste */
int ArrayList_compter(const ArrayList * liste, int valeur);

#endif
```

```
/* fichier main.c */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "arraylist.h"
#include "hashmap.h"

#define BENCHMARKS
#undef BENCHMARKS

#ifndef BENCHMARKS
#define CAPACITY 10000
#define ELEMENTS 1000000
#define RESEARCHES 10000
#else
#define CAPACITY 5
#define ELEMENTS 20
#endif

int main() {
    HashMap * map = HashMap_creer(CAPACITY);
    ArrayList * liste = ArrayList_creer(CAPACITY);
    long seed = time(NULL);
    int value;
```

```

long i;
#ifndef BENCHMARKS
    int count;
    fprintf(stderr, "Filling ArrayList\n");
    clock_t start, stop;
    start = clock();
#endif
    srand(seed);
    for(i = 0; i < ELEMENTS; ++i) {
        value = rand() % ELEMENTS;
        ArrayList_ajouter(liste, value);
    }
#endif
stop = clock();
fprintf(stderr, "Filling completed in %g s\n", (double)(stop -
    start) / CLOCKS_PER_SEC);
fprintf(stderr, "Filling HashMap\n");
start = clock();
#endif
srand(seed);
for(i = 0; i < ELEMENTS; ++i) {
    value = rand() % ELEMENTS;
    HashMap_ajouter(map, value);
}
#endif
stop = clock();
fprintf(stderr, "Filling completed in %g s\n", (double)(stop -
    start) / CLOCKS_PER_SEC);
#endif
#endif
HashMap_afficher(stdout, map);
ArrayList_afficher(stdout, liste);
#else
seed *= 42;
fprintf(stderr, "Research in ArrayList\n");
start = clock();
srand(seed);
count = 0;
for(i = 0; i < RESEARCHES; ++i) {
    value = rand() % ELEMENTS;
    count += ArrayList_compter(liste, value);
}

```

```
}

stop = clock();
fprintf(stderr, "Research completed in %g s\n", (double)(stop -
→ start) / CLOCKS_PER_SEC);
fprintf(stderr, "Counting %d elements\n", count);

fprintf(stderr, "Research in HashMap\n");
start = clock();
srand(seed);
count = 0;
for(i = 0; i < RESEARCHES; ++i) {
    value = rand() % ELEMENTS;
    count += HashMap_compteur(map, value);
}
stop = clock();
fprintf(stderr, "Research completed in %g s\n", (double)(stop -
→ start) / CLOCKS_PER_SEC);
fprintf(stderr, "Counting %d elements\n", count);
#endif
HashMap_free(&map);
ArrayList_free(&liste);
exit(EXIT_SUCCESS);
}
```

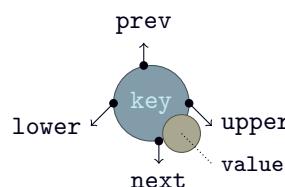
Exercice 118 (★★★ Représentation d'un lexique).

Nous vous proposons de représenter un lexique depuis la lecture de mots dans un fichier à l'aide du structure de nœuds chaînés. La structure est la suivante :

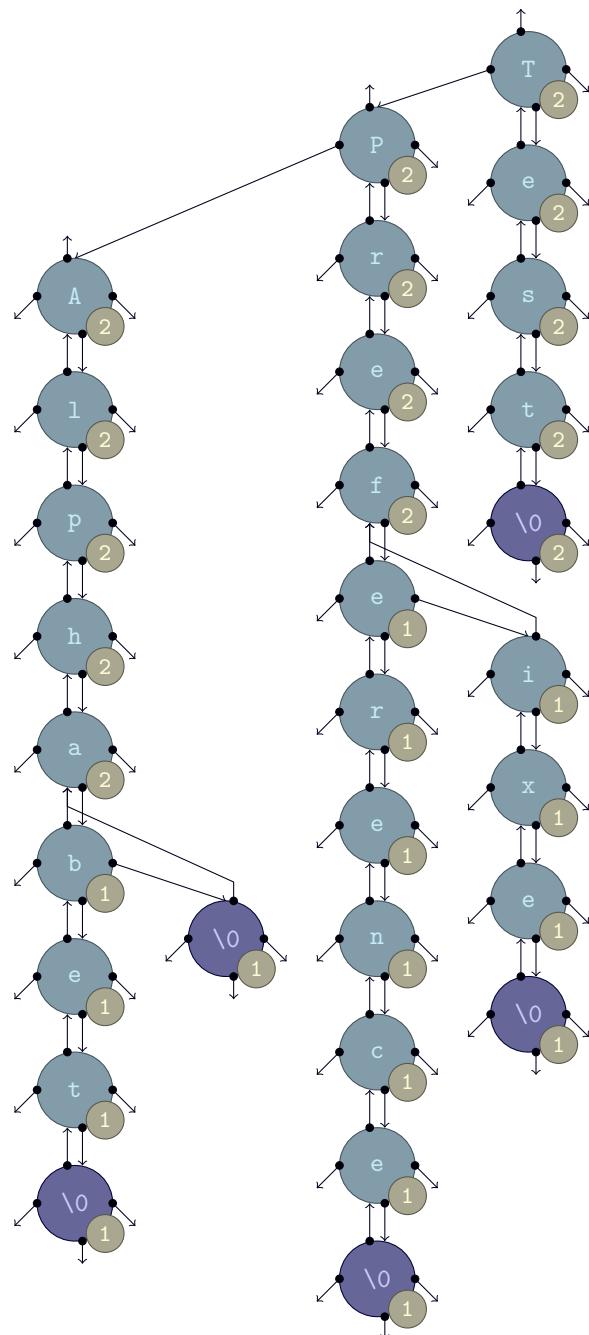
```
typedef struct Node Node;
struct Node {
    char key;
    int value;
    Node * lower;
    Node * upper;
    Node * next;
    Node * prev;
};
```

- **key** correspond à une lettre (la lettre regardée dans un mot).
- **value** correspond au nombre de fois où ce nœud a été visité avec la lettre donnée.
- **lower** correspond à un noeud vers une lettre plus petite mais au même indice dans une chaîne de caractères que le nœud courant.
- **upper** correspond à un noeud de même indice mais dont la lettre est plus grande.
- **next** correspond à un noeud dont la lettre suit la lettre courante dans un mot.
- **prev** correspond à une nœud vers la lettre qui précède celle du nœud courant.

Visuellement, on peut représenter un nœud comme suivant :



Sur les ajouts successifs des mots suivants, ceci donnerait le schéma associé ci-dessous : **Test**, **Preference**, **Prefixe**, **Test**, **Alpha** et **Alphabet**.



- Depuis cette représentation, nous afficherons les mots en ordre lexicographique :

```
Alpha : 1
Alphabet : 1
Preference : 1
Prefixe : 1
Test : 2
```

- Puis, nous afficherons le nombre d'occurrence de chaque préfixe :

```
2 : A
2 : Al
2 : Alp
2 : Alph
2 : Alpha
1 : Alphab
1 : Alphabe
1 : Alphabet
2 : P
2 : Pr
2 : Pre
2 : Pref
1 : Prefe
1 : Prefer
1 : Prefere
1 : Preferen
1 : Preferenc
1 : Preference
1 : Prefi
1 : Prefix
1 : Prefixe
2 : T
2 : Te
2 : Tes
2 : Test
```

13 Types génériques

Souvent dans le code, pour gérer plusieurs types ou des appels de différentes fonctions, on peut être obligé de réécrire des algorithmes, devoir faire des disjonctions de cas à chaque utilisation. Ceci peut vite devenir pénible. Nous verrons donc ici comment réduire cette pénibilité en langage C.

Imaginons vouloir gérer une calculatrice à laquelle on ajouterait les opérations sous forme de fonctions :

```
#define OP(name, symb, a, b) \
    printf("%s de %d et %d :\n", #name, a, b); \
    printf("%d %s %d = %d\n", a, symb, b, name(a, b));

int addition(int a, int b) { return a + b; }
int soustraction(int a, int b) { return a - b; }
int multiplication(int a, int b) { return a * b; }
int division(int a, int b) { return a / b; }
int modulo(int a, int b) { return a % b; }
int decalageGauche(int a, int b) { return a << b; }
int decalageDroite(int a, int b) { return a >> b; }
```

Une approche pour gérer la chose pourrait être par le code suivant :

```
int main() {
    int first, second;
    char op[5];
    printf(">>> ");
    scanf("%d %s %d", &first, op, &second);
    if(strcmp("+", op) == 0) {
        OP(addition, "+", first, second);
    } else if(strcmp("-", op) == 0) {
        OP(soustraction, "-", first, second);
    } else if(strcmp("*", op) == 0) {
```

```

    OP(multiplication, "*", first, second);
} else if(strcmp("//", op) == 0) {
    OP(division, "//", first, second);
} else if(strcmp("%", op) == 0) {
    OP(modulo, "%", first, second);
} else if(strcmp("<<", op) == 0) {
    OP(decalageGauche, "<<", first, second);
} else if(strcmp(">>", op) == 0) {
    OP(decalageDroite, ">>", first, second);
}
exit(EXIT_SUCCESS);
}

```

Cependant, celui-ci va être difficilement maintenable. Il faudra le modifier à chaque fois que l'on ajoute une fonction, il y a de la duplication. Peut-être que nous pourrions faire mieux avec un tableau et une structure, mais comment sauvegarder une fonction ?

13.1 Pointeurs de fonctions

Un outil pour référencer une fonction est un pointeur sur une fonction.

13.1.1 Définition

Pour fabriquer un pointeur sur un type de fonction donné, nous allons le construire comment ayant même signature (type de retour, type et nombre des arguments identique). Pour que ce soit un pointeur et non une fonction, nous mettrons le nom entre parenthèses et ajouterons une étoile. Ceci prend la syntaxe suivante :

```
typeRetour (* nom)(typeArguments);
```

Nous avons maintenant un pointeur sur des fonctions :

```
int (* operateurBinaire)(int, int) = &addition;
operateurBinaire = &soustraction;
```

En réalité pour ne pas alourdir la syntaxe des pointeurs de fonctions, le compilateur accepte de passer directement la fonction et non son adresse :

```
int (* operateurBinaire)(int, int) = addition;
operateurBinaire = soustraction;
```

13.1.2 Appel

Une appel à la fonction référencée peut se faire par déréférencement de la fonction puis l'utiliser comme on le ferait avec la fonction référencée. Bien que l'absence de déréférencement ne soit pas pénalisée par le compilateur :

```
(*operateurBinaire)(1, 1); /* valide */
operateurBinaire(1, 1);    /* autorisée */
```

Utilisation

Nous pouvons à présent maintenir une liste de nos opérations sans avoir à toucher les moments du code qui l'utilisent :

```
#define LISTOP(name, symb) {#name, #symb, name}
typedef struct OperationBinaire OperationBinaire;
struct OperationBinaire {
    char * nom;
    char * symbole;
    int (* operateur)(int, int);
};
OperationBinaire * getFromSymbole(const char * symbole,
→ OperationBinaire * liste) {
    for(; liste->nom != NULL; ++liste) {
        if(strcmp(symbole, liste->symbole) == 0) {
            return liste;
        }
    }
    return NULL;
}
```

```

int main() {
    /* Ajoutez vos opérations ici : */
    OperationBinaire operations[] = {
        LISTOP(addition, +),
        LISTOP(soustraction, -),
        LISTOP(multiplication, *),
        LISTOP(division, /),
        LISTOP(modulo, %),
        LISTOP(decalageGauche, <<),
        LISTOP(decalageDroite, >>),
        {NULL}
    };

    int first, second;
    OperationBinaire *ops = NULL;
    char op[5];
    printf(">> ");
    scanf("%d %s %d", &first, op, &second);
    if((ops = getFromSymbole(op, operations)) != NULL) {
        printf("%s de %d et %d :\n", ops->nom, first, second);
        printf("%d %s %d = %d\n", first, ops->symbole, second,
            → ops->operateur(first, second));
    }
    exit(EXIT_SUCCESS);
}

```

13.1.3 Constructions plus complexes

Il est possible de construire des types et réaliser des opérations plus complexes avec les pointeurs de fonctions. Une astuce peut être de considérer que pour travailler avec, on travaille à partir de son nom. Par exemple, si vous ajoutez des crochets derrière le nom vous pouvez déclarer un tableau statique, une étoile devant le nom donne un pointeur / tableau dynamique et ajouter des arguments derrière le nom permet de construire une fonction. Voyons comment ceci s'appliquerait pour un pointeur de fonction sur le schéma suivant :

```
typeRetour (* nom)(typeArguments);
```

Tableaux statiques

On peut construire un tableau de ce type de pointeur de fonctions :

```
typeRetour (* nom[TAILLE])(typeArguments);
```

Ceci pourrait donner par exemple le tableau suivant :

```
int (* tableau[]) (int, int) = {  
    addition,  
    soustraction,  
    NULL  
};  
  
tableau[1](2, 1);
```

Tableaux dynamiques

On peut construire un pointeur sur ce type de pointeur de fonctions :

```
typeRetour (** nom)(typeArguments);
```

Une allocation dynamique d'un tableau de pointeurs vers les fonctions vues précédemment pourrait par exemple donner le code suivant :

```
int (** pointeur)(int, int) = NULL;  
if((pointeur = (int **)(int))malloc(sizeof(int (*)(int,  
        int)) * 3)) == NULL) {  
    fprintf(stderr, "Erreur allocation.\n");  
    exit(EXIT_FAILURE);  
}  
  
*pointeur = addition;  
*(pointeur + 1) = soustraction;
```

```
*(pointeur + 2) = NULL;

(*(pointeur + 1))(2, 1);
pointeur[1](2, 1);
```

Type de retour d'une fonction

On peut construire une fonction qui renvoie ce type de pointeur de fonctions :

```
typeRetour (* nom(ArgumentsFonction))(typeArguments);
```

Ceci pourrait par exemple être utilisé pour renvoyer une fonction selon un paramètre donné comme le symbole d'une opération :

```
int (* selectOperation(const char * symbole))(int, int) {
    if(strcmp(symbole, "+") == 0)
        return addition;
    else if(strcmp(symbole, "-") == 0)
        return soustraction;
    return NULL;
}
```

13.1.4 Typedef

Il est possible de construire des types de plus en plus complexes avec les opérations vues précédemment. Cependant, ceci peut vite devenir abominable et difficile à lire. Bien que l'exemple suivant soit une allocation dynamique d'un pointeur sur une fonction qui renvoie l'adresse d'une fonction prenant en entrée un entier et renvoyant un entier. Trivial, non ?

```
int carre(int x) {
    return x * x;
}

int (* fc(int nb))(int) {
    switch(nb) {
```

```
    case 0 : return carre;
    default : return NULL;
}
}

int main() {
    int (*(** ppfc)(int))(int) = (int
        →  (**)(int))(int)malloc(sizeof(int (*)(int))(int));
    *ppfc = fc;
    printf("%d\n", (*ppfc)(0)(4));
    exit(EXIT_SUCCESS);
}
```

Tout comme pour les types vus précédemment, il est possible de créer un synonyme sur un type de pointeur de fonctions avec typedef :

```
typedef int (* mapToIntToInt)(int);

int carre(int x) {
    return x * x;
}

mapToIntToInt fc(int nb) {
    switch(nb) {
        case 0 : return carre;
        default : return NULL;
    }
}

int main() {
    mapToIntToInt (** ppfc)(int) = (mapToIntToInt
        →  (**)(int))(int)malloc(sizeof(mapToIntToInt (*)(int)));
    *ppfc = fc;
    printf("%d\n", (*ppfc)(0)(4));
    exit(EXIT_SUCCESS);
}
```

```
}
```

Ceci est aussi applicable pour simplifier le code de manière à gérer le code de manière simplifiée. Attention à bien avoir conscience des types que vous manipulez à cause de l'abstraction que ceci peut créer.

```
typedef int (* mapToIntToInt)(int);
typedef mapToIntToInt (* selectMITIFromInt)(int);

int carre(int x) {
    return x * x;
}

mapToIntToInt fc(int nb) {
    switch(nb) {
        case 0 : return carre;
        default : return NULL;
    }
}

int main() {
    selectMITIFromInt * ppfc = (selectMITIFromInt
         $\hookrightarrow$  *)malloc(sizeof(selectMITIFromInt));
    *ppfc = fc;
    printf("%d\n", (*ppfc)(0)(4));
    exit(EXIT_SUCCESS);
}
```

13.2 Pointeurs génériques

Vous avez codé votre algorithme de tri ou une autre structure de données en C. Malheureusement cet outil ne convient que pour le type pour lequel c'est paramétrisé. Vous faites copié-collé et vous modifiez les types ?

13.2.1 Intérêt

Le typage en C est une contrainte qui limite la générnicité du code. Par change, il est possible de créer des pointeurs génériques pour abstraire un type utilisé dans notre code. Ce type est le `void *`.

13.2.2 Exemple avec `qsort`

Prenons pour exemple la méthode de tri fournie par la bibliothèque standard. C'est une implémentation générique du tri rapide :

```
# man 3 qsort

QSORT(3)          Linux Programmer's Manual       QSORT(3)

NAME
    qsort, qsort_r - sort an array

SYNOPSIS
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

Pour utiliser `qsort`, vous allez devoir fournir votre tableau, le nombre d'éléments, la taille de chacun puis une fonction de comparaison. Attention comme vous pouvez le voir, cette fonction de comparaison prend en paramètres des pointeur génériques. Ceci peut par exemple se faire pour trier une liste d'entiers en ordre croissant à l'aide de l'appel à `qsort` suivant et de la fonction de comparaison suivante :

```
int intcmp(const void * firstAddress, const void * secondAddress)
→ {
    int first = *((int *)firstAddress);
    int second = *((int *)secondAddress);
    return first - second; /* first < second => first - second < 0
    → */
}

void afficherListe(int * valeurs, int taille) {
```

```

int i;
for(i = 0; i < taille; ++i) {
    if(i) printf(", ");
    printf("%d", valeurs[i]);
}
printf("\n");

int main() {
    int valeurs[] = {5, 9, 2, 0, 6, 1, 3, 8, 7, 4};
    afficherListe(valeurs, 10);
    qsort(valeurs, 10, sizeof(int), &intcmp);
    afficherListe(valeurs, 10);
    exit(EXIT_SUCCESS);
}

```

Devoir construire des fonctions qui prennent obligatoirement des arguments en pointeur générique peut être fastidieux. Pour ceci, vous pouvez utiliser le type de pointeur correspondant à votre type et effectuer un cast de votre pointeur de fonction avec la signature générique attendue pour la passer à l'outil en nécessitant :

```

int intcmp(const int * first, const int * second) {
    return *first - *second;
}

/* ... */

int main() {

    int valeurs[] = {5, 9, 2, 0, 6, 1, 3, 8, 7, 4};
    afficherListe(valeurs, 10);
    qsort(valeurs, 10, sizeof(int), (int (*)(const void *, const
        void *))&intcmp);
    afficherListe(valeurs, 10);
    exit(EXIT_SUCCESS);
}

```

{}

13.2.3 Manipulation

Vous pouvez aussi vous-même construire des pointeurs génériques pour sauvegarder une information dans vos structures de données et pour y associer des fonctionnalités. Ou comme dans le cas en `qsort` proposer des méthodes fonctionnant sur le type que vous donnera un utilisateur sous réserve qu'il vous fournisse les fonctionnalités nécessaires.

Ceci peut par exemple permettre la création d'une liste générique. Dans cette liste, l'aspect générique nous demande d'être plus vigilants sur les points suivants :

- Nous avons besoin de connaître la taille d'un élément.
- Lorsque nous avons besoin d'une fonctionnalité particulière sur un élément (affichage, allocation, libération et autres), il faut que celle-ci soit fournie lorsque nécessaire ou idéalement en amont.
- L'arithmétique des pointeurs ne se fera pas automatiquement. Une proposition connaissant le taille en octets et de travailler sur l'adresse avec des pas d'un octet. Ceci se fait par exemple avec des adresses en `unsigned char *`.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

typedef struct Liste Liste;
struct Liste {
    void * data;
    long taille;
    long capacite;
    size_t elementTaille;
    void (* elementFree)(void *);
    void (* elementPrint)(void *);
};
```

```
int Liste_create(
    Liste * liste, long capacite, size_t elementTaille,
    void (* elementFree)(void *), void (* elementPrint)(void *)) {
    liste->data = NULL;
    liste->taille = 0;
    liste->capacite = capacite;
    liste->elementTaille = elementTaille;
    liste->elementFree = elementFree;
    liste->elementPrint = elementPrint;
    if((liste->data = malloc(elementTaille * capacite)) == NULL) {
        fprintf(stderr, "Liste_create : Erreur allocation liste.\n");
        return 1; /* code d'erreur allocation */
    }
    return 0;
}

void Liste_free(Liste * liste) {
    if(liste == NULL || liste->data == NULL) {
        return;
    }
    long i;
    if(liste->elementFree) {
        for(i = 0; i < liste->taille; ++i) {
            liste->elementFree((unsigned char *) (liste->data) + (i
                * liste->elementTaille));
        }
    }
    free(liste->data);
    liste->data = NULL;
}

void Liste_afficher(Liste * liste) {
    long i;
    printf("[");
```

```
for(i = 0; i < liste->taille; ++i) {
    if(i) printf(", ");
    liste->elementPrint((unsigned char *) (liste->data) + (i
        * liste->elementTaille));
}
printf("]\n");
```

```
int Liste_ajouter(Liste * liste, void * element) {
    if(liste->taille >= liste->capacite) {
        void * tmp = NULL;
        int new_capacite = liste->capacite * 2 + 10;
        if((tmp = realloc(liste->data, new_capacite *
            liste->elementTaille)) == NULL) {
            fprintf(stderr, "Liste_ajouter : Erreur
                reallocation.\n");
            return 1; /* code d'erreur allocation */
        }
        liste->data = tmp;
        liste->capacite = new_capacite;
    }
    memcpy((unsigned char *) (liste->data) + (liste->taille *
        liste->elementTaille), element, liste->elementTaille);
    ++(liste->taille);
    return 0;
}

void intPrint(int * value) {
    printf("%d", *value);
}

int main() {

    Liste intListe;
```

```

if(Liste_create(&intListe, 0, sizeof(int), NULL, (void (*)(void
→  *))&intPrint)) {
    fprintf(stderr, "Erreur allocation intListe :
→  arrêt.\n");
    exit(EXIT_FAILURE);
}
int valeur;
printf("Entrez des valeurs positives : ");
scanf("%d", &valeur);
while(valeur >= 0) {
    Liste_ajouter(&intListe, &valeur);
    scanf("%d", &valeur);
}
Liste_afficher(&intListe);
Liste_free(&intListe);

exit(EXIT_SUCCESS);
}

```

13.3 Fonctions variadiques

Nous avons jusqu'ici toujours défini des fonctions avec un nombre de paramètres connu. Cependant `printf` et `scanf` prennent bien un nombre d'argument variable, non ?

En effet, il est possible de fabriquer des fonctions avec un nombre d'arguments variables à l'aide de la bibliothèque `stdarg.h`. Celle-ci fournit un type `va_list` qui permet de gérer les arguments variables d'une fonction. Une fonction variadique s'organise de la manière suivante :

```

typeRetour fonctionVariadique(parametres, dernierParametre, ...) {
    va_list ap; /* argument pointer */
    va_start(ap, dernierParametre); /* initialisation */
    /* Traitement et récupération avec va_arg(ap, typeElement) */
    va_end(ap); /* arrêt d'utilisation */

```

```
}
```

Pour définir une fonction qui calcule une moyenne, ceci pourrait par exemple donner le code suivant :

```
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>

double moyenne(int nombre, ...) {
    double somme = 0;
    int i;
    va_list ap;
    va_start(ap, nombre);
    for(i = 0; i < nombre; ++i) {
        somme += va_arg(ap, double);
    }
    va_end(ap);
    if(nombre)
        somme /= nombre;
    return somme;
}

int main() {
    printf("%g\n", moyenne(3, 10., 11., 13.));
    exit(EXIT_SUCCESS);
}
```

13.4 Bibliothèques dynamiques

Jusqu'ici, nous avons utilisé des fonctionnalités de la bibliothèque standard par exemple. Mais tout comme celles-ci ont été codées par d'autres, vous pouvez coder vos propres bibliothèques en langage C et les utiliser.

Un exemple sur une machine Linux est `/lib/x86_64-linux-gnu/libc.so.6` correspondant aux fonctionnalités standard compilées du langage C :

```
# nm -D /lib/x86_64-linux-gnu/libc.so.6
000000000004f9d0 T a64l
000000000000406b0 T abort
000000000003ecd20 B __abort_msg
00000000000438d0 T abs
00000000000122560 T accept
00000000000122f70 T accept4
000000000001101f0 W access
00000000000116e80 T acct
000000000001181a0 W addmnttent
00000000000051e30 W addseverity
...
0000000000015bc40 T xencrypt
0000000000010f800 T __xmknod
0000000000010f860 T __xmknodat
00000000000051450 T __xpg_basename
000000000003f4f0 W __xpg_sigpause
000000000000a83b0 T __xpg_strerror_r
000000000001597c0 T xpri_register
000000000001598f0 T xpri_unregister
0000000000010f710 T __xstat
0000000000010f710 T __xstat64
```

En général, en langage C, nous séparons les fichiers d'entête .h dans un répertoire `include`. Par exemple `/usr/include` sur une machine Linux où vous pourriez trouver les fichiers d'entête des déclarations de la bibliothèque standard comme `stdlib.h`.

13.4.1 Compiler sa bibliothèque

Pour créer votre bibliothèque, séparez votre code en deux répertoires :

- **include** : pour les fichiers d'entêtes .h.
- **src** : pour l'implémentation de vos fonctionnalités dans des fichiers .c.

Prenons l'exemple d'une bibliothèque **esgi** définie par les sources suivantes :

```
/* +-----+ */
/* | FICHIER : include/esgilib.h | */
/* +-----+ */

#ifndef DEF_HEADER_ESGI_LIB
#define DEF_HEADER_ESGI_LIB

void ESGI_init();

void ESGI_quit();

#endif
```

```
/* +-----+ */
/* | FICHIER : src/esgilib.c | */
/* +-----+ */

#include "../include/esgilib.h"

#include <stdio.h>

void ESGI_init() {
    printf("ESGI start\n");
}

void ESGI_quit() {
    printf("ESGI stop\n");
}
```

Les fichiers source .c doivent être compilés avec l'option **-fPIC** (sauts relatifs dans le code assembleur plutôt qu'absolus). Les fichiers objets doivent ensuite être linkés avec l'option **-shared** pour en faire une bibliothèque partagée.

```
gcc -c src/esgilib.c -fPIC
gcc -o libesgi.so esgilib.o -shared
```

Il est ensuite possible d'inspecter la table des symboles de la bibliothèque :

```
# nm -D libesgi.so
00000000000201028 B __bss_start
          w __cxa_finalize
00000000000201028 D _edata
00000000000201030 B _end
00000000000000062a T ESGI_init
00000000000000063d T ESGI_quit
000000000000000650 T _fini
          w __gmon_start__
000000000000000508 T _init
          w _ITM_deregisterTMCloneTable
          w _ITM_registerTMCloneTable
          U puts
```

13.4.2 Utiliser sa bibliothèque à la compilation

Par défaut, le compilateur va rechercher les bibliothèques dans `/usr/lib` et les fichiers d'entête dans `/usr/include`. Dans le cas où vous souhaiteriez utiliser une bibliothèque personnalisée, vous devrez indiquer le chemin où trouver vos bibliothèques. Ceci peut se faire avec l'option `-L`. De même pour indiquer un répertoire personnalisé où trouver vos fichiers d'entête vous l'option `-I` est disponible. Tout comme les autres bibliothèques vous devrez indiquer son utilisation au linker par le `-l`.

Pour utiliser notre bibliothèque `libesgi.so`, nous pourrions utiliser le code source suivant :

```
/* +-----+ */
/* | FICHIER : src/main.c | */
/* +-----+ */
#include <esgilib.h>
/* Possible à l'aide du -I./include */
#include <stdlib.h>
```

```
int main() {
    ESGI_init();
    ESGI_quit();
    exit(EXIT_SUCCESS);
}
```

Puis la compilation suivante :

```
gcc -o prog src/main.c -L. -I./include -lesgi
```

Ceci donnerait ensuite l'exécution suivante :

```
# ./prog
ESGI start
ESGI stop
```

À noter qu'il est possible de visualiser les bibliothèques utilisées par votre programme par la commande `ldd` :

```
# ldd prog
    linux-vdso.so.1 (0x00007ffe72bfb000)
    libesgi.so (0x00007f73712a9000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
                  (0x00007f7370eb8000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f73716ad000)
```

13.4.3 Charger sa bibliothèque à l'exécution

À noter qu'une utilisation de vos bibliothèques n'est pas obligatoirement à relier au linkage. En effet, vos bibliothèques partagées peuvent être chargées à l'exécution à l'aide de la bibliothèque de chargement de bibliothèques dynamiques d'entête `dlopen.h`. Ceci permet par exemple d'avoir un système de plugins pour votre application sans avoir à la recompiler.

Pour utiliser dynamiquement la bibliothèque partagée, il faudra l'ouvrir via `dlopen` et la fermer lorsque les symboles récupérés ne seront plus utilisés dans le

programme via `dlclose`. Un symbole pour être récupéré via le retour de `dlsym` sauvegardé dans un pointeur de fonction compatible.

À noter lors de l'ouverture de la bibliothèque par `dlopen` que les options suivantes :

- `RTLD_NOW` précharge tous les symboles de la bibliothèque à l'ouverture.
- `RTLD_LAZY` charge le symbole uniquement lorsqu'il est récupéré.

Pour charger dynamiquement la bibliothèque proposée précédemment, nous pourrions utiliser le code suivant :

```
/* +-----+ */
/* / FICHIER : src/main.c / */
/* +-----+ */

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

int main() {
    const char * lib_path = "libesgi.so";
    void (*ESGI_init)() = NULL;
    void (*ESGI_quit)() = NULL;

    void * dlptra = NULL;
    if((dlptr = dlopen(lib_path, RTLD_NOW)) == NULL) {
        fprintf(stderr, "Erreur d'ouverture du plugin \"%s\"\n", lib_path, dlerror());
    }
    if((ESGI_init = (void (*)())dlsym(dlptra, "ESGI_init")) == NULL)
    {
        fprintf(stderr, "Erreur de lecture de \"ESGI_init\" du plugin\n"
                    "\"%s\"\n", lib_path, dlerror());
        exit(EXIT_FAILURE);
    }
    if((ESGI_quit = (void (*)())dlsym(dlptra, "ESGI_quit")) == NULL)
    {
        dlclose(dlptra);
    }
}
```

```
fprintf(stderr, "Erreur de lecture de \\"ESGI_quit\\" du plugin  
→ \"%s\\n%s\\n\", lib_path, dlerror());  
exit(EXIT_FAILURE);  
}  
  
ESGI_init();  
ESGI_quit();  
  
dlclose(dlptra);  
dlptr = NULL;  
exit(EXIT_SUCCESS);  
}
```

Celui-ci peut être compilé et lancé via les commandes suivantes (-ldl est à ajouter au linkage) :

```
# gcc -o prog src/main.c -ldl  
# ./prog  
ESGI start  
ESGI stop
```

13.5 Résumé

Une variable qui peut référencer une fonction est modélisée par un pointeur de fonction :

```
typeRetour (* pointeur)(typeParametres); /* pointeur sur une
→ fonction de signature typeRetour fonction(typeParametres) */
pointeur = fonction; /* affectation */
pointeur(arguments); /* appel à fonction via pointeur */
```

Il est possible de considérer un pointeur de fonction comme un type de variable pour y appliquer les opérations suivantes :

```
typeRetour (* tableau[])(typeParametres); /* tableau de pointeurs
→ de fonctions */
typeRetour (** pointeur)(typeParametres); /* pointeur sur un
→ pointeur de fonction */
typeRetour (* fonction(parametresDeFonction))(typeParametres); /*
→ signature d'une fonction renvoyant un pointeur de fonction */
typedef typeRetour (* synonyme)(typeParametres); /* synonyme sur
→ un pointeur de fonction */
```

Lorsque l'on souhaite manipuler des données de manière générique indépendamment de leur type, il est possible d'utiliser un pointeur générique `void *` :

```
void * pointeurGenerique; /* pointeur générique pour sauvegarde
→ d'information */
(type *)pointeurGenerique; /* transformation du pointeur générique
→ en le type réellement enregistré */
void * fonctionGenerique(void * pointeur); /* fonction utilisable
→ dans un code générique */
```

Il est possible de définir une fonction prenant un nombre variable de paramètres à l'aide de la bibliothèque `stdarg` :

```
typeRetour fonctionVariadique(parametres, dernierParametre, ...)
{
    va_list ap; /* argument pointer */
    va_start(ap, dernierParametre); /* initialisation */
    /* Traitement et récupération avec va_arg(ap, typeElement) */
    va_end(ap); /* arrêt d'utilisation */
}
```

Il est possible de créer sa propre bibliothèque, la relier à son programme à la compilation ou à l'exécution à l'aide des options de compilation suivantes :

```
-fPIC # Préparation d'un fichier .c en fichier .o compatible pour
      → une bibliothèque partagée
-shared # Linkage de fichiers .o dans une bibliothèque partagée
-L[CHEMIN] # Indiquer un répertoire où trouver des bibliothèques
-I[CHEMIN] # Indiquer un répertoire où trouver des fichiers
      → d'entête
-l[LIBNAME] # Linker sa bibliothèque lib[LIBNAME].so
-ldl # Bibliothèque pour chargement de code à l'exécution
```

Pour charger une bibliothèque à l'exécution il est possible d'utiliser les fonctionnalités suivantes de `dlopen.h` :

```
/* Chargement d'une bibliothèque partagée (NULL si échec) : */
void * dlopen(const char * path, int flag);
/* flag peut être RTLD_NOW ou RTLD_LAZY par exemple. */
/* Libération d'une bibliothèque partagée : */
int dlclose(void * dlptra);
/* Chargement d'un symbole de la bibliothèque (NULL si échec) :
   → */
void * dlsym(void * dlptra, const char * symbol_name);
/* Détail de l'erreur si échec : */
char * dlerror();
```

13.6 Entraînement

Exercice 119 (★★★ Trier une liste de points).

Proposer un programme qui trie une liste de points :

1. Par rapport à leur distance à l'origine.
2. Par rapport à un point donné.

```
Distance origine :  
(0, 1) 1  
(-3, 0) 3  
(0, 3) 3  
(2, 3) 3.60555  
(4, 0) 4  
Distance (3, 3) :  
(2, 3) 1  
(0, 3) 3  
(4, 0) 3.16228  
(0, 1) 3.60555  
(-3, 0) 6.7082
```

Exercice 120 (★★★★ Benchmark de qsort).

1. Implémenter un tri par tas générique.
2. Proposer un programme qui compare les performances du tri par tas avec celles de `qsort` de la bibliothèque standard `stdlib`.
3. Que penser des performances de `qsort` ?

Exercice 121 (★★★ Trier des pointeurs de fonctions).

Votre binôme est à deux doigts de ragequit le cours de langage C. Il essaie de trier des pointeurs de fonctions mais rien ne marche. Vous devez lui venir en aide ! Les pointeurs de fonctions seraient triés par ordre croissant de la valeur qu'ils renverraient pour une valeur donnée.

Code de votre binôme :

```
#include <stdio.h>
#include <stdlib.h>

float carre(float x) {
    return x * x;
}

float cube(float x) {
    return x * x * x;
}

float inverse(float x) {
    return 1.f / x;
}

float oppose(float x) {
    return -x;
}

int fcmp(float first(float x), float second(float x)) {
    return first(x) - second(x);
}

int main() {
    float fonctions(float x)[] = {
        carre,
        cube,
        inverse,
        oppose
    };
    float targets[] = {
        -2.f,
        -0.5f,
    }
}
```

```

    0.5f,
    2.f
};

int i, j;
for(i = 0; i < 4; ++i) {
    qsort(fonctions(targets[i]), sizeof(float f(float x)), 4,
        → fcmp);
    printf("for %g :\n", targets[i]);
    for(j = 0; j < 4; ++j) {
        printf(" - %s(%g) = %g\n", fonctions(targets[i])[j].name,
            → targets[i], fonctions(targets[i])[j]);
    }
}
exit(EXIT_SUCCESS);
}

```

Sortie attendue par le professeur :

```

for -2 :
- cube(-2) = -8
- inverse(-2) = -0.5
- oppose(-2) = 2
- carre(-2) = 4
for -0.5 :
- inverse(-0.5) = -2
- cube(-0.5) = -0.125
- carre(-0.5) = 0.25
- oppose(-0.5) = 0.5
for 0.5 :
- oppose(0.5) = -0.5
- cube(0.5) = 0.125
- carre(0.5) = 0.25
- inverse(0.5) = 2
for 2 :
- oppose(2) = -2
- inverse(2) = 0.5
- carre(2) = 4
- cube(2) = 8

```

Exercice 122 (★★★ Changer de langue).

Proposer un programme qui permettra à l'avenir de proposer différents langages sans avoir à modifier le code de base. Ceci fonctionnerait en indiquant la bibliothèque du langage correspondant en argument. Vous devrez compléter le code suivant :

```
/* +-----+ */
/* / FICHIER : main.c / */
/* +-----+ */
#include <stdio.h>
#include <stdlib.h>

typedef struct Langage Langage;
struct Langage {
    void (*saluer)();
    void (*quit)();
};

int main(int argc, char * argv[]) {
    const char * lang_path = argc > 1 ? argv[1] : "fr.so";
    Langage lang = Langage_chargeur(lang_path);
    lang.saluer();
    lang.quit();
    Langage_liberer(&lang);
    exit(EXIT_SUCCESS);
}
```

Votre code doit permettre de charger dynamiquement les codes suivants (sans les compiler avec l'exécutable : ils doivent pouvoir se compiler sous forme de bibliothèque et fournir des .iso :

```
$ ./executable fr.so
Chargement de la langue depuis "fr.so"
Bonjour !
Au revoir.
$ ./executable en.so
Chargement de la langue depuis "en.so"
Hello !
Bye.
```

Les codes correspondants sont les suivants :

```
/* +-----+ */
/* / FICHIER : fr.c / */
/* +-----+ */
#include <stdio.h>

void saluer() {
    printf("Bonjour !\n");
}

void quit() {
    printf("Au revoir.\n");
}
```

```
/* +-----+ */
/* / FICHIER : en.c / */
/* +-----+ */
#include <stdio.h>

void saluer() {
    printf("Hello !\n");
}

void quit() {
    printf("Bye.\n");
}
```

Exercice 123 (★★★★★ Simulation de combattants génériques).

Proposer l'implémentation des fichiers d'entête suivants. Vous devrez proposer le chargement dynamique de personnages sur base du modèle donné et pouvant utiliser les fonctionnalités de votre exécutable. Le but est que chaque personnage possède 4 capacités (codées ou non dans la bibliothèque dynamique) dont il en choisit une à utiliser par tour dans un combat automatique.

```
/* +-----+ */
/* | FICHIER : Personnage.h | */
/* +-----+ */

#ifndef DEF_HEADER_PERSONNAGE
#define DEF_HEADER_PERSONNAGE

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

typedef struct Stats Stats;
struct Stats {
    int vie; /* vie */
    int atk; /* attaque */
    int def; /* defense */
    int vit; /* vitesse */
};

typedef struct Personnage Personnage;
struct Personnage {
    char nom[21]; /* nom du personnage. */
    Stats start; /* statistiques initiales. */
    Stats current; /* statistiques courrantes. */
    void (*coups[4])(Personnage *, Personnage *); /* capacités du
        → personnage. */
    int (*choix)(const Personnage *, const Personnage *); /* choix
        → de la capacité selon la configuration. */
    void * dlptr; /* référence vers bibliothèque dynamique associée.
        → */
};

/* Charge un personnage depuis un fichier .so */
Personnage Personnage_chargeur(const char * path);
```

```

/* Replace les valeurs courantes en fonction des valeurs */
/* initiales si nécessaire. */
void Personnage_keep_in_bounds(Personnage * perso);

/* Initialise le personnage à ses stats initiaux. */
void Personnage_init(Personnage * perso);

/* Ferme la bibliothèque dynamique ouverte. */
void Personnage_quit(Personnage * perso);

/* Affiche le personnage. */
void Personnage_display(const Personnage * perso);

/* Indique si le personnage est vivant. */
int Personnage_est_vivant(const Personnage * perso);

/* Donne le nom du personnage. */
const char * Personnage_nom(const Personnage * perso);

/* Indique si le personnage est plus rapide que l'autre. */
int Personnage_est_plus_rapide(const Personnage * self, const
→ Personnage * other);

/* Effectue l'action associée au personnage. */
void Personnage_action(Personnage * self, Personnage * other);

/* Donne un soldat simple. */
Personnage Personnage_default_soldier();

/* Donne un garde. */
Personnage Personnage_defensive_soldier();

#endif

```

```

/* +-----+ */
/* | FICHIER : Spell.h | */
/* +-----+ */
#ifndef DEF_HEADER_SPELL
#define DEF_HEADER_SPELL

```

```
#include "Personnage.h"

void Spell_coup_simple(Personnage * self, Personnage * other);

void Spell_se_soigner(Personnage * self, Personnage * other);

void Spell_boost_atk(Personnage * self, Personnage * other);

void Spell_boost_def(Personnage * self, Personnage * other);

#endif
```

```
/* +-----+ */
/* | FICHIER : AI.h | */
/* +-----+ */
#ifndef DEF_HEADER_AI
#define DEF_HEADER_AI

#include "Personnage.h"

/* Priorise d'attaquer l'adversaire. */
int AI_hit_prior(const Personnage * self, const Personnage *
→ other);

/* Priorise de se booster avant de combattre. */
int AI_buff_prior(const Personnage * self, const Personnage *
→ other);

#endif
```

Le code doit permettre d'importer ses personnages comme codé dans le fichier suivant :

```
/* +-----+ */
/* | FICHIER : plugins/src/Balourd.c | */
/* +-----+ */
#include "../../include/Personnage.h"
#include "../../include/Spell.h"
#include "../../include/AI.h"
```

```
#include <math.h>

void Spell_coup_lourd(Personnage * self, Personnage * other) {
    int deg = pow((float)(self->current.atk) / other->current.def,
    ↪ 2.f);
    if(deg <= 0) {
        deg = 1;
    }
    other->current.vie -= deg;
    printf("%s frappe %s et lui inflige %d dégats.\n", self->nom,
    ↪ other->nom, deg);
    Personnage_keep_in_bounds(other);
}

int AI_bouriner(const Personnage * self, const Personnage * other)
↪ {
    if(rand() % 3 == 0) {
        return 2;
    }
    if(other->current.vie < 0.20 * other->start.vie) {
        return 0;
    }
    if(self->current.vie < 0.50 * self->start.vie) {
        return 1;
    }
    return 0;
}

Personnage Personnage_instantiate() {
    return (Personnage) {
        .nom = "Balourd",
        .start = (Stats) {
            .vie = 120,
            .atk = 100,
            .def = 150,
            .vit = 6
        },
        .coups = {
            &Spell_coup_lourd,
            &Spell_se_soigner,
            &Spell_boost_atk,
        }
    };
}
```

```
    &Spell_boost_def  
},  
.choix = &AI_bouriner  
};  
}
```

Exercice 124 (★★★★★ HashMap et ArrayList génériques).

Oscar salue vos aptitudes en langage C pour être arrivé jusqu'ici, mais il a un dernier défi pour vous avant de passer au projet final. Il a pu observer des outils génériques dans d'autres langages de programmation. Il vous propose d'essayer de coder ce type de structures en langage C. L'idée serait de proposer une table de hachage qui puisse prendre des entrées génériques : comme des listes génériques. À vous de proposer une implémentation qui réponde à cette demande et soit en mesure de modéliser la sortie suivante :

```
HashMap<string, ArrayList> : {"grands" : ArrayList<long> :  
→ [5000000000, 30000000000, -42000000000], "caracteres" :  
→ ArrayList<char> : ['E', 'S', 'G', 'I'], "entiers" :  
→ ArrayList<int> : [1, 17, 3, 42, 5], "a virgule" :  
→ ArrayList<float> : [4.5, 13.37, 1.2, 8.1, 99.9, 4.2, 1.2],  
→ "texte" : ArrayList<string> : ["Hello", "ESGI"]}
```

Exercice 125 (★★★★ Tableaux dynamiques à n dimensions).

Oscar est ravi que vous lisiez cet exercice, car il a un défi pour vous ! En l'occurrence, il est assez simple de construire des tableaux statiques à nombre de dimensions quelconque en Langage C. Cependant Oscar aimeraient inventer le `nmalloc` : une fonction d'allocation d'un tableau dynamique à n dimensions : libérable entièrement avec un seul `free` bien sûr.

Il a commencé à vous proposer des exemples d'utilisation de la fonction :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <assert.h>

void * nmalloc(size_t atomic, int dimensions, ...) {
    /* TODO : construire un tableau dynamique à 'dimensions' dimensions. */
    /* (Chaque élément fait atomic octets). */
    int i;
    va_list ap;
    va_start(ap, dimensions);
    for(i = 0; i < dimensions; ++i) {
        printf("Dimensions %d / %d = %d\n", i + 1, dimensions, va_arg(ap,
            → int));
    }
    va_end(ap);
    return NULL;
}

void test_dim4() {
    int nx = 4, ny = 5, nz = 6, nw = 7;
    int **** voxel = nmalloc(sizeof(int), 4, nx, ny, nz, nw);
    int i, j, k, l;
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
            for(k = 0; k < nz; ++k) {
                for(l = 0; l < nw; ++l) {
                    voxel[i][j][k][l] = i * ny * nz * nw + j * nz * nw + k * nw +
                        → l;
                }
            }
        }
    }
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
```

```
    for(k = 0; k < nz; ++k) {
        for(l = 0; l < nw; ++l) {
            assert(voxel[i][j][k][l] == i * ny * nz * nw + j * nz * nw + k
                   * nw + l);
        }
    }
}
free(voxel);
voxel = NULL;
}

void test_voxel() {
    int nx = 5, ny = 7, nz = 10;
    int *** voxel = nmalloc(sizeof(int), 3, nx, ny, nz);
    int i, j, k;
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
            for(k = 0; k < nz; ++k) {
                voxel[i][j][k] = i * ny * nz + j * nz + k;
            }
        }
    }
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
            for(k = 0; k < nz; ++k) {
                assert(voxel[i][j][k] == i * ny * nz + j * nz + k);
            }
        }
    }
    free(voxel);
    voxel = NULL;
}

void test_pixel() {
    int nx = 10, ny = 20;
    int ** pixel = nmalloc(sizeof(int), 2, nx, ny);
    int i, j;
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
            pixel[i][j] = i * ny + j;
        }
    }
    for(i = 0; i < nx; ++i) {
        for(j = 0; j < ny; ++j) {
```

```
        assert(pixel[i][j] == i * ny + j);
    }
}
free(pixel);
pixel = NULL;
}

void test_list() {
    int nx = 10;
    int * list = nmalloc(sizeof(int), 1, nx);
    int i;
    for(i = 0; i < nx; ++i) {
        list[i] = i;
    }
    for(i = 0; i < nx; ++i) {
        assert(list[i] == i);
    }
    free(list);
    list = NULL;
}

int main() {
    test_list();
    test_pixel();
    test_voxel();
    test_dim4();
    printf("Pas d'arrêt, vérifier avec valgrind.\n");
    exit(EXIT_SUCCESS);
}
```

Exercice 126 (∞ Algorithme d'exponentiation générique). Un scientifique a choisi le Langage C pour ses performances comparé au langage Python. Cependant, il doit recoder chaque algorithme à chaque fois qu'il souhaite changer de méthode opératoire, là où c'était bien plus magique avec Python. Il a donc besoin de votre talent pour réussir à gérer des algorithmes de manière générique. Il vous indique de tester avec l'algorithme de l'exponentiation rapide pour les éléments suivants :

- Réels : utilisation d'un type `double`.
- Entiers modulaires : utilisation du type `unsigned int` où les opérations sont réalisées modulo un entier p .
- Entiers : utilisation du type `mpz_t` (de la bibliothèque GMP pour des entiers sans limite).
- Matrices 2x2 : utilisation d'opérations matricielles pour un type donné (précédemment ou un type Matrice 2x2).

À noter qu'Oscar est passé par là et dit qu'avec une structure donnant les informations suivantes pour un type, ce ne serait pas si compliqué à implémenter :

- Taille du type en mémoire.
- Récupération du neutre de l'addition (zéro).
- Récupération du neutre de la multiplication (un ou identité).
- Opération d'addition.
- Opération de multiplication.
- Copie d'un élément.
- Opposé d'un élément par l'addition.
- Inverse d'un élément par la multiplication (lorsque possible).
- Affichage d'un élément.
- Données supplémentaires (valeur modulaire, système opératoire).

Comme preuve de concept, le scientifique aimerait que votre méthode fonctionne sur les exemples suivants, les calculant en moins d'une milliseconde chacun par l'exponentiation rapide :

1. **Réel**, exemples :

$$3^{10} = 59\ 049$$

$$2^{-3} = 0.125$$

2. **Entiers modulaires**, exemples :

$$3^{30} \equiv 260 [1337]$$

$$42^{125} \equiv 124 [127]$$

3. **Entiers sans limite**, exemple :

$$2^{10\ 000} = 19950631\dots96709376 \text{ (3 011 chiffres)}$$

4. **Matrice 2x2 réelle**, exemples :

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{1\ 000} \simeq \begin{pmatrix} 7.03304 \times 10^{208} & 4.34666 \times 10^{208} \\ 4.34666 \times 10^{208} & 2.68638 \times 10^{208} \end{pmatrix}$$

5. **Matrice 2x2 d'entiers modulaires**, exemples :

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^{42} \equiv \begin{pmatrix} 90 & 49 \\ 10 & 100 \end{pmatrix} [127]$$

6. **Matrice 2x2 d'entiers sans limite**, exemple :

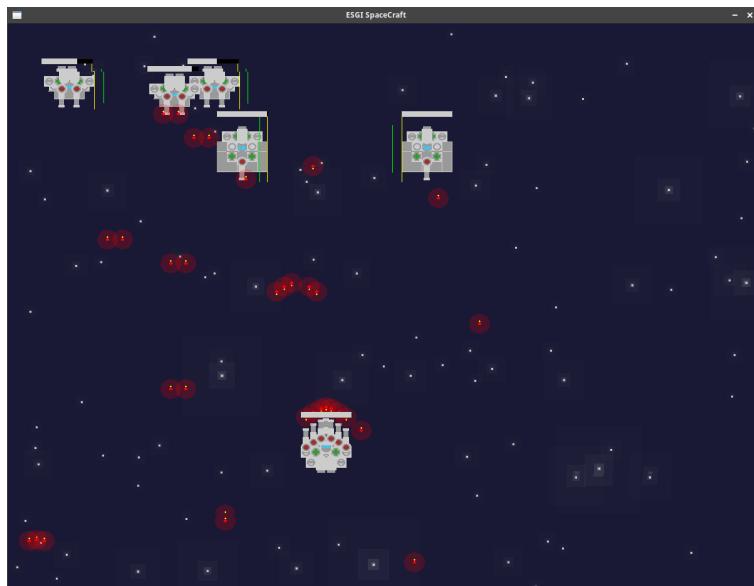
$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{19\ 999} = \begin{pmatrix} 25311623\dots13093125 & 15643443\dots11617501 \\ 15643443\dots11617501 & 96681797\dots01475624 \end{pmatrix} \text{ (4 180 chiffres)}$$

7. Matrice 2×2 de Matrices 2×2 d'entiers modulaires, exemple :

$$\begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} & \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix} \\ \begin{pmatrix} 9 & 10 \\ 13 & 14 \end{pmatrix} & \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix} \end{pmatrix}^{42} \equiv \begin{pmatrix} \begin{pmatrix} 115 & 102 \\ 102 & 13 \end{pmatrix} & \begin{pmatrix} 89 & 76 \\ 51 & 89 \end{pmatrix} \\ \begin{pmatrix} 89 & 51 \\ 76 & 89 \end{pmatrix} & \begin{pmatrix} 13 & 102 \\ 102 & 115 \end{pmatrix} \end{pmatrix} [127]$$

Exercice 127 (∞ Space invaders modulaire). Oscar a commencé un projet de jeu, mais ne s'en sort plus. Il met sa fierté de côté pour vous demander de l'aide. Il veut créer un Space Invaders-like mais en plus modulable. C'est-à-dire qu'il pourrait proposer des vaisseaux et composants de vaisseaux en plugins (qu'un joueur pourrait coder en Langage C pour en ajouter).

Robert a devancé Oscar et vous a montré ce à quoi pourrait ressembler un tel jeu, il vous a même fourni ses fichiers plugins :



Oscar imagine le concept suivant :

- Le joueur peut contrôler un vaisseau au clavier.
- De adversaires affrontent le joueur.
- Chaque vaisseau est décrit en Langage C comme un plugin.
- Chaque vaisseau est un ensemble de composants (ses statistiques sont la sommes des statistiques de chaque composant).
- Chaque composant est décrit en Langage C comme un plugin.
- Un composant peut apporter des statistiques ou une action (génération d'un projectile).
- La manipulation vaisseau du joueur se ferait au clavier.

- Les vaisseaux seraient soumis à une pseudo simulation physique (le contrôle d'un vaisseau affecte un vecteur de déplacement qui est ajouté à la position).
- Les composants seraient les suivants :
 - Cockpit : base de points attribuables pour construire le vaisseau.
 - Propulseurs : force avant, arrière, latérale (droite et gauche).
 - Canons : générateurs de projectiles.
 - Armure : protection du vaisseau.
 - Utilitaires : non-catégorisés (comme la réparation).

Proposez un jeu modulaire dans cette idée, à vous de jouer !

Voici le code qu'Oscar a commencé :

```
/* src/main.c */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>
#include <Game.h>
#include <Component.h>

/* Paramètres de la fenêtre : */
const int largeur = 1200;
const int hauteur = 900;
const char * titre = "ESGI SpaceCraft";
Game game;

void affichage(SDL_Surface * ecran) {
    unsigned long t = SDL_GetTicks();
    int x, y, i;
    /* Remplissage de l'écran par un gris foncé uniforme : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 25, 25, 53));
    for(i = 1; i < 100; ++i) {
        x = 123 * i * i + 800 * sin((t + i * i * 100.) / 20000.);
        y = 753 * i + t / 4. + 100 * cos((t + i * 100.) / 2000.);
        x = ((x % largeur) + largeur) % largeur;
        y = ((y % hauteur) + hauteur) % hauteur;
        boxRGBA(ecran, x - 1, y - 1, x + 1, y + 1, 204, 204, 204, 255);
    }
}
```

```

}

int main(int argc, char * argv[]) {
    Ship * ship = Ship_load((argc > 1) ? argv[1] :
    ↵ "extensions/Ship_basic.so", 1, Position_create(largeur / 2, 3 *
    ↵ hauteur / 4));
    game = Game_create(largeur, hauteur, ship);
    List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
    ↵ "extensions/Ship_basic.so", 2, Position_create(largeur / 2, hauteur
    ↵ / 8)));
    List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
    ↵ "extensions/Ship_basic.so", 2, Position_create(largeur / 4, hauteur
    ↵ / 9)));
    List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
    ↵ "extensions/Ship_basic.so", 2, Position_create(3 * largeur / 4,
    ↵ hauteur / 9)));
    List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
    ↵ "extensions/Ship_tank.so", 2, Position_create(largeur / 4, 2 *
    ↵ hauteur / 9)));
    List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
    ↵ "extensions/Ship_tank.so", 2, Position_create(3 * largeur / 4, 2 *
    ↵ hauteur / 9)));
    srand(time(NULL));
    /* Cr ation d'une fen tre SDL : */
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE |
    ↵ SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n", SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);

    int active = 1;
    SDL_Event event;

    while(active) {

        affichage(ecran);
        Game_draw(&game, ecran);
        SDL_Flip(ecran);
    }
}

```

```
while(SDL_PollEvent(&event)) {  
  
    switch(event.type) {  
        /* Utilisateur clique sur la croix de la fenêtre : */  
        case SDL_QUIT : {  
            active = 0;  
        } break;  
  
        /* Utilisateur enfonce une touche du clavier : */  
        case SDL_KEYDOWN : {  
            switch(event.key.keysym.sym) {  
                /* Touche Echap : */  
                case SDLK_ESCAPE : {  
                    active = 0;  
                } break;  
  
                case SDLK_SPACE : {  
                    game.player->control.ship_action = 1;  
                } break;  
  
                case SDLK_z :  
                case SDLK_UP : {  
                    game.player->control.move_forward = 1;  
                } break;  
  
                case SDLK_s :  
                case SDLK_DOWN : {  
                    game.player->control.move_backward = 1;  
                } break;  
  
                case SDLK_q :  
                case SDLK_LEFT : {  
                    game.player->control.move_left = 1;  
                } break;  
  
                case SDLK_d :  
                case SDLK_RIGHT : {  
                    game.player->control.move_right = 1;  
                } break;  
  
                default : break;  
            }  
        } break;
```

```

    case SDL_KEYUP : {
        switch(event.key.keysym.sym) {
            case SDLK_SPACE : {
                game.player->control.ship_action = 0;
            } break;

            case SDLK_z :
            case SDLK_UP : {
                game.player->control.move_forward = 0;
            } break;

            case SDLK_s :
            case SDLK_DOWN : {
                game.player->control.move_backward = 0;
            } break;

            case SDLK_q :
            case SDLK_LEFT : {
                game.player->control.move_left = 0;
            } break;

            case SDLK_d :
            case SDLK_RIGHT : {
                game.player->control.move_right = 0;
            } break;

            default : break;
        }
    } break;

    default : break;
}

int delay = SDL_GetTicks() + 1000 / 60;
Game_play_step(&game);
delay -= SDL_GetTicks();
if(delay > 0) {
    SDL_Delay(delay);
}
}

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}

```

Il a essayé de déterminer quelques entêtes depuis les plugins de Robert :

```
/* include/Stats.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_STATS
#define DEF_HEADER_ESGI_SPACE_CRAFT_STATS

#include <stdio.h>

typedef struct Stats Stats;
struct Stats {
    double forward_force;
    double backward_force;
    double right_force;
    double left_force;
    double mass;
    double health;
    double armor;
    double damage;
    double repair;
    int points;
};

Stats Stats_zero();

Stats Stats_base(double mass, double health, double armor, int points);

#endif
```

```
/* include/List.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_LIST
#define DEF_HEADER_ESGI_SPACE_CRAFT_LIST

#include <stdlib.h>

typedef struct List * List;
struct List {
    /* TODO */
};

List List_empty();
```

```

void List_free(List * list);

List List_alloc(void * value, List next);

void * List_add(List * list, void * value);

void * List_remove(List * list, void * value);

void List_FOREACH(List list, void (*process)(void *));

void * List_predicate(List list, int (*predicate)(void *));

#endif
```

```

/* include/Projectile.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_PROJECTILE
#define DEF_HEADER_ESGI_SPACE_CRAFT_PROJECTILE

#include "Position.h"
#include <SDL/SDL.h>

struct Game;
struct Ship;

typedef struct Projectile Projectile;
struct Projectile {
    Position position;
    int projectileId;
    int used;
    int (*action)(Projectile *, struct Ship *, struct Game *);
    void (*draw)(Projectile *, SDL_Surface *, int, int, int, int);
    void * userData;
};

Projectile * Projectile_alloc(Position position);

#endif
```

```

/* include/Component.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_COMPONENT
#define DEF_HEADER_ESGI_SPACE_CRAFT_COMPONENT
```

```
#include "Stats.h"
#include "Position.h"
#include "Projectile.h"
#include <SDL/SDL.h>

struct Ship;
struct Game;

typedef enum {
    CT_NONE = 0,
    CT_COCKPIT,
    CT_ARMOR,
    CT_WEAPON,
    CT_PULSOR,
    CT.Utility
} ComponentType;

typedef struct Component Component;
struct Component {
    ComponentType type;
    Stats stats;
    int width;
    int height;
    int componentId;
    Position position;
    void (*action)(Component *, struct Ship *, struct Game *);
    void (*draw)(Component *, SDL_Surface *, int, int, int, int);
    void * userData;
    void * dlptr;
};

Component * Component_alloc(ComponentType type, Stats stats,
                           int width, int height, Position position);

void Component_free(Component * component);

Component * Component_load(const char * path, Position position);

#endif
```

```
/* include/Ship.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_SHIP
```

```
#define DEF_HEADER_ESGI_SPACE_CRAFT_SHIP

#include "Stats.h"
#include "Position.h"
#include "List.h"
#include "Component.h"

#define SHIP_SCALE 8

typedef struct Controler Controler;
struct Controler {
    int move_forward;
    int move_backward;
    int move_left;
    int move_right;
    int ship_action;
    int reload_action;
};

struct Game;

typedef struct Ship Ship;
struct Ship {
    Stats stats;
    int width;
    int height;
    int team;
    double health;
    int shipId;
    Position position;
    Position moving;
    List components;
    List projectiles;
    Controler control;
    void * userData;
    void * dlp;
};

Controler Controler_create();

Ship * Ship_alloc(int team, Position position);

Ship * Ship_load(const char * path, int team, Position position);

int Ship_attack(Projectile * projectile, double range, Ship * ship);
```

```
void Ship_apply_damages(double damages, Ship * ship);

#endif
```

```
/* include/Game.h */
#ifndef DEF_HEADER_ESGI_SPACE_CRAFT_GAME
#define DEF_HEADER_ESGI_SPACE_CRAFT_GAME

#include "Ship.h"
#include "Projectile.h"
#include <SDL/SDL.h>

typedef struct Game Game;
struct Game {
    int width;
    int height;
    Ship * player;
    List opponents;
};

Game Game_create(int width, int height, Ship * player);

void Game_free(Game * game);

void Game_draw(Game * game, SDL_Surface * ecran);

void Game_play_step(Game * game);

#endif
```

Les fichiers plugins de Robert pourraient ressembler à ceci :

```
/* plugins/Component_canon.c */
#include <Component.h>
#include <Ship.h>
#include <Game.h>

#include <SDL/SDL.h>
#include <SDL/SDL_gfxPrimitives.h>

void Projectile_draw(Projectile * projectile, SDL_Surface * ecran, int x,
    int y, int s, int t) {
```

```

int offset = 1;
if(t > 1) {
    offset = -1;
}
filledCircleRGBA(ecran, x, y, 2. * s, 255, 0, 0, 53);
filledCircleRGBA(ecran, x, y, 0.25 * s, 255, 0, 0, 255);
filledCircleRGBA(ecran, x, y + 0.5 * s * offset, 0.125 * s, 255, 255,
    ↵ 0, 255);
}

int Projectile_action(Projectile * projectile, Ship * ship, Game * game)
{
    if(projectile->used || projectile->position.y < 0 ||
    ↵ projectile->position.y > game->height) {
        projectile->used = 1;
        return 0;
    }
    int offset = 1;
    if(ship->team > 1) {
        offset = -1;
    }
    projectile->position.y -= offset * 10.;
    if(ship->team > 1) {
        if(Ship_attack(projectile, 1., game->player)) {
            Ship_apply_damages(200., game->player);
        }
    } else {
        void process(Ship * ship) {
            if(projectile->used) {
                return;
            }
            if(Ship_attack(projectile, 1., ship)) {
                Ship_apply_damages(200., ship);
            }
        };
        List_FOREACH(game->opponents, (void (*)(void *))process);
    }
    return 1;
}

Projectile * Projectile_instantiate(Position position) {
    Projectile * projectile = Projectile_alloc(position);
    projectile->action = Projectile_action;
    projectile->draw = Projectile_draw;
    return projectile;
}

```

```
}

void draw(Component * component, SDL_Surface * ecran, int x, int y, int
→ s, int t) {
    int offset = 1;
    if(t > 1) {
        offset = -1;
    }
    filledCircleRGBA(ecran, x, y + offset * s, s, 204, 204, 204, 255);
    boxRGBA(ecran, x - 0.4 * s, y - s - 1.5 * offset * s, x + 0.4 * s, y +
→ s - 1.5 * offset * s, 153, 153, 153, 255);
    boxRGBA(ecran, x - 0.6 * s, y - s, x + 0.6 * s, y + s, 204, 204, 204,
→ 255);
    boxRGBA(ecran, x - 0.6 * s, y - 0.2 * s - 2.5 * offset * s, x + 0.6 *
→ s, y + 0.2 * s - 2.5 * offset * s, 204, 204, 204, 255);
    filledCircleRGBA(ecran, x, y + offset * s, 0.8 * s, 153, 153, 153,
→ 255);
    filledCircleRGBA(ecran, x, y + offset * s, 0.5 * s, 153, 51, 51, 255);
}

void action(Component * component, Ship * ship, Game * game) {
    int offset = 1;
    if(ship->team > 1) {
        offset = -1;
    }
    List_add(&(ship->projectiles), Projectile_instantiate(Position_create(
        ship->position.x + offset * SHIP_SCALE * component->position.x,
        ship->position.y + offset * SHIP_SCALE * component->position.y
    )));
}

Component * Component_instantiate(Position position) {
    Component * component = Component_alloc(
        CT_WEAPON,
        Stats_base(50., 20., 10., -1),
        1, 3,
        position
    );
    component->stats.damage = 200.;
    component->draw = draw;
    component->action = action;
    return component;
}
```

```

/* plugins/Ship_basic.c */
#include <Ship.h>

Ship * Ship_instantiate(int team, Position position) {
    Ship * ship = Ship_alloc(
        team,
        position
    );
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_cannon.so",
    ↳ Position_create(1.5, -1.5)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_cannon.so",
    ↳ Position_create(-1.5, -1.5));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_backpulsor.so",
    ↳ Position_create(-1., 2.5)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_backpulsor.so",
    ↳ Position_create(1., 2.5)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_repair.so",
    ↳ Position_create(2., 1.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_repair.so",
    ↳ Position_create(-2., 1.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_direction.so",
    ↳ Position_create(4., 1.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_direction.so",
    ↳ Position_create(-4., 1.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_brakes.so",
    ↳ Position_create(3., 0.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_brakes.so",
    ↳ Position_create(-3., 0.)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_armor.so",
    ↳ Position_create(0., -1.5)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_armor.so",
    ↳ Position_create(2., -1.5)));
    List_add(&(ship->components),
    ↳ Component_load("extensions/Component_armor.so",
    ↳ Position_create(-2., -1.5)));

```

```
List_add(&(ship->components),  
    Component_load("extensions/Component_cockpit.so",  
    Position_create(0., 0.)));  
return ship;  
}
```

14 Bonus : Opérations bit-à-bit

Nous avons vu ensemble que la machine travaille sous une représentation binaire et que ceci pouvait être affiché en hexadécimal. Cependant, est-il possible de travailler directement avec cette représentation binaire sans bidouiller avec nos opérateurs arithmétiques ?

C'est possible avec des opérateurs dits bit-à-bit. Cependant ces concepts seront étudiés en 3ème année à l'ESGI Paris, ceci est donc un cadeau pour le lecteur.

14.1 Décalages

Un premier opérateur est l'opérateur de décalage. Celui-ci permet de décaler la représentation binaire vers la droite ou vers la gauche dans la capacité de son type.

14.1.1 À gauche

Le décalage à gauche de x par la valeur d décale les bits de d chiffres vers la gauche dans la représentation binaire de x et place des zéros aux emplacements vides. Utiliser `<<` est équivalent à multiplier x par 2^d . À noter que s'il y a dépassement de capacité, les bits sont perdus.

12	0	0	0	0	1	1	0	0
$12 \ll 2 = 48$	0	0	1	1	0	0	0	0

Il est par exemple possible de l'utiliser pour construire les puissances de 2 :

```
int i;
unsigned int valeur;
for(i = 0; i < 31; ++i) {
    valeur = ((unsigned int)1 << i);
    printf("%12u | 0x%08X | %2de bit à 1\n", valeur, valeur, i + 1);
}
```

Par défaut le 1 que l'on décale est en `int`. Pour s'éviter quelques désagrément, un conseil peut être de le caster en le type que l'on souhaite récupérer :

```
unsigned long x;
x = (1 << 42);
printf("%lu, le bit 42 n'existe pas ? On m'a menti !\n", x);
x = ((unsigned long)1 << 42);
printf("%lu, ouf, rassuré !\n", x);
```

14.1.2 À droite

L'opérateur de décalage à droite `>>` fonctionne de la même manière que l'opérateur de décalage à gauche.

24	0	0	0	1	1	0	0	0
<code>24 >> 3 = 3</code>	0	0	0	0	0	0	1	1

14.2 Négation

Il est possible de nier chaque bit à l'aide de l'opérateur de négation bit-à-bit `~`.

7	0	0	0	0	0	1	1	1
<code>~7</code>	1	1	1	1	1	0	0	0

À noter que la machine utilise cet opérateur de négation bit-à-bit notamment pour se représenter des nombres négatifs. En effet, la machine utilise le complément à deux pour représenter des nombres négatifs : négation bit-à-bit puis ajout de 1. On peut le vérifier :

```
int i = 42;
printf("%d + %d = %d\n", i, ~i + 1, i + ~i + 1);
```

Cet opérateur peut être combiné avec les opérateurs de décalage pour obtenir un nombre pour lequel il manque une puissance de 2 (soit un 1 dans la représentation binaire).

```

int i;
unsigned int valeur;
for(i = 0; i < 31; ++i) {
    valeur = ~((unsigned int)1 << i);
    printf("%12u | 0x%08X | sans %2de bit à 1\n",
           valeur, valeur, i
           + 1);
}

```

À noter que l'opérateur de négatif fonctionne dans la plage de valeurs du type sur lequel il opère :

```

unsigned long x;
unsigned int nombre = 1;
x = ~(nombre << 2);
printf("%lu, Il y a un soucis ?\n", x);
x = ~((unsigned long)nombre << 2);
printf("%lu, Effectivement, c'est un peu plus long en fait !\n",
       x);

```

14.3 Opérateurs booléens

Nous avons vu précédemment les opérateurs booléens qui permettent d'opérer sur des valeurs de vérité. Cependant pour certaines applications, il peut être intéressant d'en avoir une version bit-à-bit.

14.3.1 et

L'opérateur d'intersection `&&` permet de vérifier que deux valeur sont vraies. L'opérateur d'intersection bit-à-bit `&` permet d'allumer un bit correspond à 1 si ceux des deux opérandes le sont et le laisser à 0 sinon.

3	0	0	1	1
6	0	1	1	0
2	0	0	1	0

Pour gérer un système d'options, on pourrait définir une variable par option et vérifier si chacune est vraie. Cependant, une alternative proposée est de le faire dans une seule variable et appliquer un masque à l'aide de l'opérateur `&` pour vérifier que le bit correspondant est bien présent dans la valeur fournie. Pour lister ces options, on peut utiliser un type énuméré qui offrira plus de lisibilité ensuite dans le code.

```
typedef enum {
    OPTION1 = 0x0001,
    OPTION2 = 0x0002,
    OPTION3 = 0x0004
} Options;

int main() {
    Options valeur = OPTION2 + OPTION3;
    if(valeur & OPTION1)
        printf("option 1\n");
    if(valeur & OPTION2)
        printf("option 2\n");
    if(valeur & OPTION3)
        printf("option 3\n");
    exit(EXIT_SUCCESS);
}
```

14.3.2 ou inclusif

L'opérateur de réunion `||` permet de déterminer si au moins l'une des deux valeurs données en opérande est vraie. Son équivalent bit-à-bit `|` procède de même sur les bits des deux opérandes pour une nouvelle valeur entière.

3	0	0	1	1
6	0	1	1	0
7	0	1	1	1

L'opérateur de réunion bit-à-bit peut être utilisé pour gérer des options. Ceci permet d'assembler différentes options dans une même valeur sans chevauchements tels qu'il pourrait y avoir avec l'addition.

```
Options first = OPTION1 | OPTION2;
Options second = OPTION2 | OPTION3;
Options result = first | second;
if(result & OPTION1)
    printf("option 1\n");
if(result & OPTION2)
    printf("option 2\n");
if(result & OPTION3)
    printf("option 3\n");
```

14.3.3 ou exclusif : xor

L'opérateur de réunion exclusive \sim aussi nommé "Xor" permet de vérifier qu'au moins l'une des opérande est vraie mais qu'il n'y a pas d'intersection : les deux ne sont pas vraies en même temps. Il est possible de l'utiliser sur des valeurs de vérité (ceci demandera potentiellement l'ajout de parenthèses pour les expressions) bien qu'il n'existe qu'en version bit-à-bit.

3	0	0	1	1
6	0	1	1	0
5	0	1	0	1

Il peut par exemple être utilisé pour récupérer les options qui ne sont pas communes entre deux opérandes :

```
Options first = OPTION1 | OPTION2;
Options second = OPTION2 | OPTION3;
Options result = first ^ second;
if(result & OPTION1)
    printf("option 1\n");
```

```
if(result & OPTION2)
    printf("option 2\n");
if(result & OPTION3)
    printf("option 3\n");
```

En algorithmique, nous avons vu l'échange de valeurs à l'aide d'une variable temporaire. Cependant, l'échange de deux entiers peut se faire sans variable temporaire avec un Xor.

```
int a = 42, b = 1337;
a ^= b; /* a ^ b */
b ^= a; /* b ^ a ^ b = a */
a ^= b; /* a ^ b ^ a = b */
```

14.4 Résumé

Il est possible d'opérer directement sur les bit à l'aide d'opérations bit-à-bit :

- Décalage bit-à-bit à gauche :

```
entier << decalage;
/* renvoie la valeur de entier décalée de decalage vers la
   ↵ gauche */
/* En binaire : */
/* 00000110 */
/* <<      2 */
/* = 00011000 */
```

- Décalage bit-à-bit à droite :

```
entier >> decalage;
/* renvoie la valeur de entier décalée de decalage vers la
   ↵ droite */
/* En binaire : */
/* 00000110 */
/* >>      2 */
/* = 00000001 */
```

- Négation bit-à-bit :

```
~entier;
/* renvoie la valeur de entier avec renversement des valeurs
   ↵ des bits */
/* En binaire : */
/* ~ 00000110 */
/* = 11111001 */
```

- Intersection bit-à-bit :

```
first & second;
/* renvoie un entier où seuls les bits allumés dans first et
   ↳ second sont conservés */
/* En binaire : */
/* 00000110 */
/* & 00000011 */
/* = 00000010 */
```

- Union inclusive bit-à-bit :

```
first | second;
/* renvoie un entier où les bits allumés dans au moins first
   ↳ ou second sont conservés */
/* En binaire : */
/* 00000110 */
/* | 00000011 */
/* = 00000111 */
```

- Union exclusive bit-à-bit (Xor) :

```
first ^ second;
/* renvoie un entier où les bits allumés dans first ou second
   ↳ sans l'être dans les deux sont conservés */
/* En binaire : */
/* 00000110 */
/* ^ 00000011 */
/* = 00000101 */
```

14.5 Entraînement

Exercice 128 (** Application opérations bit-à-bit).

Compléter le code suivant en utilisant les opérations bit-à-bit donnant la sortie ci-dessous :

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    unsigned char entier8;
    unsigned int entier32;
    unsigned int entier32bis;
    unsigned long entier64;
    int test;

    /* TODO : entier32 15e bit à 1, autres à 0 */
    printf("%08x (15e bit à 1)\n", entier32);
    /* TODO : entier32 14e bit à 0, autres à 1 */
    printf("%08x (14e bit à 0)\n", entier32);

    /* TODO : entier64 43e bit à 1, autres à 0 */
    printf("%016lx (43e bit à 1)\n", entier64);
    /* TODO : entier64 3e bit à 0, autres à 1 */
    printf("%016lx (3e bit à 0)\n", entier64);

    /* TODO : entier32 13e et 1e bit à 1, autres à 0 */
    printf("%08x (13e et 12e bit à 1)\n", entier32);

    /* TODO : entier32 mettre le 12e bit à 0 sans changer les autres
     * bits */
    printf("%08x (12e bit à 0)\n", entier32);

    /* TODO : entier32 changer le 13e bit sans changer les autres
     * bits */
    printf("%08x (changement 13e bit)\n", entier32);

    entier8 = 0x1;
    /* TODO : entier32 affecter au 11e bit la valeur de entier8 */
    printf("%08x (affectation 11e bit)\n", entier32);
```

```

/* TODO : mettre vrai dans test si le 11e bit de entier32 est 1
   ↳ */
printf("%08x (test 11e bit : %d)\n", entier32, test);

entier32 = 0xFF;
/* TODO : mettre vrai dans test si les 3 bits de poids faible de
   ↳ entier32 sont 1 */
printf("%08x (test des 3 bits de poids faible à 1 : %d)\n",
   ↳ entier32, test);

entier32 = 0xF0;
/* TODO : mettre vrai dans test si les 4 bits de poids faible de
   ↳ entier32 sont 0 */
printf("%08x (test des 4 bits de poids faible à 0 : %d)\n",
   ↳ entier32, test);

entier32 = 0FFE80F12;
entier32bis = 0x0017FOED;

/* TODO : mettre vrai dans test si les bits de entier32 et
   ↳ entier32bis sont différents */
printf("%08x et %08x (test bits tous différents : %d)\n",
   ↳ entier32, entier32bis, test);

exit(EXIT_SUCCESS);
}

```

```

00004000 (15e bit à 1)
fffffdfff (14e bit à 0)
0000040000000000 (43e bit à 1)
fffffffffffffb (3e bit à 0)
00001800 (13e et 12e bit à 1)
00001000 (12e bit à 0)
00000000 (changement 13e bit)
00000400 (affectation 11e bit)
00000400 (test 11e bit : 1)
000000ff (test des 3 bits de poids faible à 1 : 1)
000000f0 (test des 4 bits de poids faible à 0 : 0)
ffe80f12 et 0017f0ed (test bits tous différents : 1)

```

Exercice 129 (★★★ Gérer une grille sur un entier).

Implémentez les fonctionnalités qui permettent de gérer une grille d'éléments présents ou absents sur un entier long :

```
#include <stdio.h>
#include <stdlib.h>

typedef unsigned long Bit8x8Grid;

Bit8x8Grid Bit8x8Grid_creer();

int Bit8x8Grid_getoffset(int x, int y);

void Bit8x8Grid_afficher(FILE * flow, Bit8x8Grid grille);

void Bit8x8Grid_set(Bit8x8Grid * grille, int x, int y, unsigned
→ char valeur);

unsigned char Bit8x8Grid_get(Bit8x8Grid grille, int x, int y);

int main() {
    Bit8x8Grid grille = Bit8x8Grid_creer();
    Bit8x8Grid_set(&grille, 3, 1, 1);
    Bit8x8Grid_set(&grille, 7, 7, 1);
    Bit8x8Grid_afficher(stdout, grille);
    exit(EXIT_SUCCESS);
}
```

```
~~~~~  
~~~#~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~  
~~~~~#
```

Exercice 130 (★★★ Buffer pour lecture et écriture bit-à-bit).

Nous avons vu précédemment comment écrire octet par octet dans un fichier et effectuer des opérations bit-à-bit dans des variables. Cependant la taille d'une variable est limitée et nous aimerais être capable d'avoir une structure de données dans laquelle écrire et lire en bit-à-bit pour potentielle utilisation avec un fichier. Implémentez une telle structure. Celle-ci devrait fonctionner avec le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include "bitbuffer.h"

#define VERBOSE
#undef VERBOSE

int main() {
    BitBuffer buffer = BitBuffer_creer();
    unsigned long infos[] = {
        1, 0x1,
        5, 0xf,
        16, 0xffff,
        32, 0x18181818,
        5, 0x7,
        16, 0xffff,
        32, 0x18181818,
        5, 0xf,
        1, 0x1,
        1, 0x1,
        1, 0x0,
        1, 0x1,
        1, 0x0,
        1, 0x0,
        1, 0x1,
        16, 0xffff,
        31, 0x8181818,
        5, 0x7,
        31, 0x8181818,
        5, 0xf,
        0
    };
    int i;
    for(i = 0; infos[2 * i] > 0; ++i) {
```

```
BitBuffer_write(&buffer, infos[2 * i + 1], infos[2 * i]);  
#ifdef VERBOSE  
    BitBuffer_print(stderr, &buffer);  
#endif  
}  
  
unsigned long data = 0;  
for(i = 0; infos[2 * i] > 0; ++i) {  
    BitBuffer_read(&buffer, &data, infos[2 * i]);  
#ifdef VERBOSE  
    BitBuffer_print(stderr, &buffer);  
#endif  
    printf("(1) %lx\n", data == infos[2 * i + 1], data);  
}  
BitBuffer_free(&buffer);  
exit(EXIT_SUCCESS);  
}
```

et afficher le résultat suivant :

```
(1) 1  
(1) f  
(1) ffff  
(1) 18181818  
(1) 7  
(1) ffff  
(1) 18181818  
(1) f  
(1) 1  
(1) 1  
(1) 0  
(1) 1  
(1) 0  
(1) 0  
(1) 1  
(1) ffff  
(1) 8181818  
(1) 7  
(1) 8181818  
(1) f
```


Cinquième partie

**Bonus : jouons rapidement
avec SDL 1.2**

Table des matières

15 Initialisation	493
15.1 Préambule	493
15.2 Mise en place	493
15.2.1 Linux	493
15.2.2 Windows	495
15.2.3 Mac	498
15.3 Fenêtre	500
15.3.1 SDL Init	500
15.3.2 SDL SetVideoMode	500
16 Affichage	503
16.1 Dessiner un rectangle	504
16.1.1 SDL FillRect	504
16.1.2 SDL Rect	505
16.2 Créer une surface	506
16.2.1 SDL CreateRGBSurface	506
16.2.2 SDL BlitSurface	508
16.3 Images en BMP	510
16.3.1 SDL Image	513
16.3.2 Dessiner une sous-image	514
16.4 Résumé	517
17 Événements	519
17.1 Récupération d'événements	519
17.1.1 SDL WaitEvent	519
17.1.2 SDL PollEvent	520
17.2 Analyse d'événements	520
17.2.1 Types d'événements	520

17.2.2 Clavier	521
17.2.3 Boutons de souris	523
17.2.4 Coordonnées de souris	523
17.3 Résumé	526
18 Améliorations graphiques : SDL GFX	529
18.1 Dessiner des primitives	529
18.1.1 pixel	529
18.1.2 hline	530
18.1.3 vline	530
18.1.4 rectangle	531
18.1.5 roundedRectangle	532
18.1.6 line	533
18.1.7 circle	534
18.1.8 ellipse	536
18.1.9 arc	537
18.1.10 trigon	539
18.1.11 polygon	540
18.1.12 texturedPolygon	542
18.1.13 bezier	543
18.2 Écrire du texte	544
18.2.1 character	544
18.2.2 string	545
18.3 Faire tourner une image	546
18.3.1 rotozoomSurface	546
18.3.2 zoomSurface	547
18.4 Quelques usages de SDL_imageFilter	548
18.4.1 Références	549
18.4.2 SDL_imageFilterAdd	549
18.4.3 SDL_imageFilterMean	550
18.4.4 SDL_imageFilterSub	550
18.4.5 SDL_imageFilterAbsDiff	551
18.4.6 SDL_imageFilterAddByte	551
18.4.7 SDL_imageFilterSubByte	552
18.4.8 SDL_imageFilterMultByByte	552
18.4.9 SDL_imageFilterBinarizeUsingThreshold	553
18.4.10 SDL_imageFilterClipToRange	553

19 Texte : SDL TTF	555
19.1 Mise en place	555
19.2 Ouverture d'un Font	555
19.3 Générer une image depuis un texte	556
19.4 Effets de style	558
20 Musique : SDL Mixer	561
20.1 Mise en place	561
20.2 Jouer une musique	563
20.2.1 Chargement d'une musique	563
20.2.2 Lecture	563
20.2.3 Volume	563
20.2.4 Fin de musique	564
20.2.5 Contrôler la musique	564
20.3 Canaux	565
20.3.1 Mise en place	565
20.3.2 Jouer des sons	565
20.3.3 Paramétrage spatial	567
21 Parallélisme : SDL Thread	569
21.1 Mise en place	569
21.2 Lancer un thread	569
21.3 Partager des ressources	570
21.4 Gestion parallèle d'événements, d'affichage et d'une simulation	573
22 Réseau : SDL Net	577
22.1 Mise en place	577
22.2 Avec TCP	577
22.2.1 Initialisation	577
22.2.2 Initialisation serveur	578
22.2.3 Initialisation client	578
22.2.4 Acceptation d'un client par le serveur	579
22.2.5 Échange d'informations	580
22.3 Exemple d'utilisation pour un mini-jeux	580

15 Initialisation

15.1 Préambule

Cette partie sur SDL fait suite des demandes d'étudiants, elle est donc facultative et à étudier en autonomie. La partie SDL peut être codée dans des travaux proposés en première année et demandée comme utilisation d'une bibliothèque externe en seconde année (avantage : vous avez une aide sur cette bibliothèque). Amusez vous !

15.2 Mise en place

15.2.1 Linux

Sous Linux, vous pouvez installer SDL 1.2 à l'aide des commandes suivantes :

```
sudo apt-get install libsdl1.2-dev
sudo apt-get install libsdl-image1.2-dev
sudo apt-get install libsdl-gfx1.2-dev
sudo apt-get install libsdl-ttf2.0-dev
sudo apt-get install libsdl-mixer1.2-dev
sudo apt-get install libglib2.0-dev
```

La compilation avec SDL se fait en ajoutant `-lSDL` et les bibliothèques que vous utiliseriez :

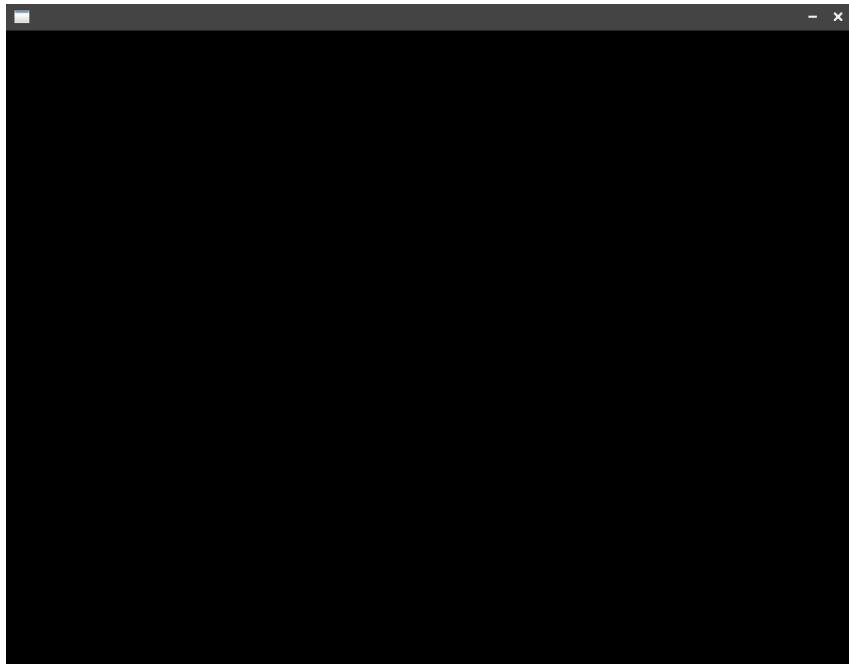
```
gcc main.c -lSDL
gcc main.c -lSDL -lSDL_image -lSDL_ttf -lSDL_gfx -lSDL_mixer
```

Vous pouvez vérifier la bonne installation de SDL en compilant le code suivant. Ce code lance une fenêtre sur fond noir et peut se fermer en cliquant sur la croix en haut :

```
#include <SDL/SDL.h>

int main() {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Surface * ecran = SDL_SetVideoMode(800, 600, 32,
        → SDL_HWSURFACE);
    SDL_Event event;
    int active = 1;
    while(active) {
        SDL_WaitEvent(&event);
        switch(event.type) {
            case SDL_QUIT : active = 0; break;
        }
    }
    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

L'exécution donne le résultat suivant :

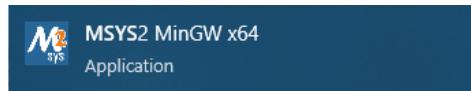


15.2.2 Windows

Sous Windows, vous pouvez installer SDL à l'aide des commandes suivantes :

```
pacman -S mingw-w64-x86_64-SDL
pacman -S mingw-w64-x86_64-SDL_image
pacman -S mingw-w64-x86_64-SDL_ttf
pacman -S mingw-w64-x86_64-SDL_gfx
pacman -S mingw-w64-x86_64-SDL_mixer
```

À noter que vous devrez passer sous **MSYS2 MinGW x64** pour compiler (là où a été installée SDL).



La compilation sous MSYS2 avec SDL se fait en ajoutant `-lmingw32 -lSDLmain -lSDL` et les bibliothèques que vous utiliserez :

```
gcc main.c -lmingw32 -lSDLmain -lSDL
gcc main.c -lmingw32 -lSDLmain -lSDL -lSDL_image -lSDL_ttf
→ -lSDL_gfx -lSDL_mixer
```

Compiler sous Windows avec MSYS2 vous demandera d'ajouter une définition en entête avant inclusion de SDL et d'avoir une signature complète pour la fonction `main` :

```
/* Nécessaire à l'utilisation de SDL via MSYS2 : */
/* (demande modification de la signature de main) */
#define SDL_MAIN_HANDLED
#include <SDL/SDL.h>

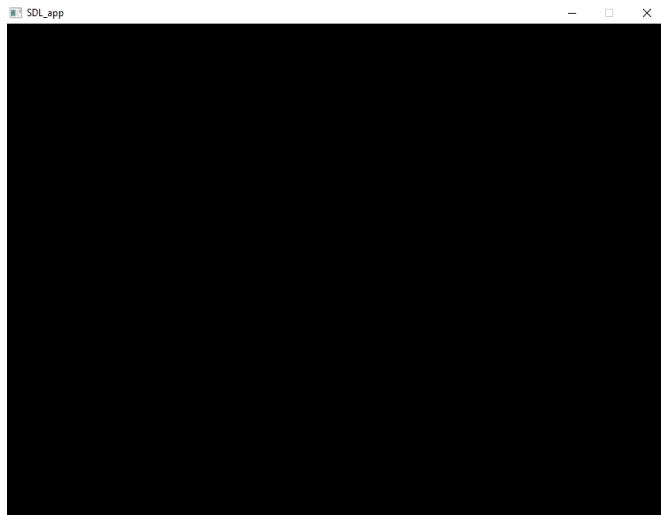
/* main doit être donnée dans sa forme complète : */
int main(int argc, char * argv[]) {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Surface * ecran = SDL_SetVideoMode(800, 600, 32,
→     SDL_HWSURFACE);
    SDL_Event event;
    int active = 1;
```

```

while(active) {
    SDL_WaitEvent(&event);
    switch(event.type) {
        case SDL_QUIT : active = 0; break;
    }
}
SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}

```

L'exécution donne le résultat suivant :



À noter que si vous souhaitez partager votre application sous Windows, vous devrez copier les fichiers .dll de la bibliothèque SDL (dans le dossier C:\msys64\mingw64\bin) à côté de votre application :

Nom	Modifié le	Type	Taille
libSDL_gfx-16.dll	30/11/2018 11:52	Extension de l'app...	92 Ko
libSDL_image-1-2-0.dll	17/12/2022 09:11	Extension de l'app...	119 Ko
libSDL_mixer-1-2-0.dll	09/08/2021 06:17	Extension de l'app...	200 Ko
main	24/10/2023 17:31	Fichier C	5 Ko
prog	24/10/2023 17:33	Application	267 Ko
SDL.dll	14/03/2022 16:00	Extension de l'app...	413 Ko
SDL_ttf.dll	08/05/2021 23:59	Extension de l'app...	32 Ko

Problème d'installation de l'entête de SDL GFX

Pour des raisons obscures, l'entête de SDL GFX installée peut être invalide. Il faudra remplacer l'inclusion de `SDL.h` par `SDL/SDL.h` (puisque l'entête de SDL est dans un dossier dans le dossier `include`) :

```
M /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
ooooo@DESKTOP-MOPOESN MINGW64 ~
$ cd "C:/Users/oooo/Downloads/S1_CC1/start_04_shooter"
ooooo@DESKTOP-MOPOESN MINGW64 /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
$ gcc -o prog main.c -lmingw32 -lSDLmain -lSDL -lSDL_gfx
In file included from main.c:7:
C:/msys64/mingw64/include/SDL/SDL_gfxPrimitives.h:38:10: fatal error: SDL.h: No such file or directory
   38 | #include <SDL.h>
      |          ^~~~~~
compilation terminated.

ooooo@DESKTOP-MOPOESN MINGW64 /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
$ nano "C:/msys64/mingw64/include/SDL/SDL_gfxPrimitives.h"
ooooo@DESKTOP-MOPOESN MINGW64 /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
$ gcc -o prog main.c -lmingw32 -lSDLmain -lSDL -lSDL_gfx
ooooo@DESKTOP-MOPOESN MINGW64 /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
$
```

Le fichier d'entête `C:/msys64/mingw64/include/SDL/SDL_gfxPrimitives.h` peut être édité avec votre IDE et directement via `nano` dans le terminal (Ctrl+x et Ctrl+o pour sauvegarder et fermer) :

```
M /c/Users/oooo/Downloads/S1_CC1/start_04_shooter
GNU nano 7.2          C:/msys64/mingw64/include/SDL/SDL_gfxPrimitives.h
3. This notice may not be removed or altered from any source
distribution.

Andreas Schiffler -- aschiffler at ferzkopp dot net
*/
#ifndef _SDL_gfxPrimitives_h
#define _SDL_gfxPrimitives_h

#include <math.h>
#ifndef M_PI
#define M_PI    3.1415926535897932384626433832795
#endif

#include <SDL/SDL.h>

/* Set up for C function definitions, even when using C++ */
#ifndef __cplusplus
extern "C" {
#endif

AG Help      A0 Write Out  A1 Where Is  A2 Cut      A3 Execute  AC Location  M-U Undo
A0 Exit      A0 Read File A1 Replace  A0 Paste     A1 Justify  A1 Go To Line M-E Redo
```

15.2.3 Mac

Concernant Mac, la démarche d'installation reprend celle proposée pour l'installation de la bibliothèque [MLV](#).

Plus explicitement, vous pouvez suivre les instructions suivantes :

Installation SDL

Pour l'installation de la base de SDL 1.2, le paquet correspondant est [SDL-1.2.15.dmg](#). Son installation se fait ensuite par les commandes suivantes :

```
hdiutil attach SDL-1.2.15.dmg
sudo cp -r /Volumes/SDL/SDL.framework /Library/Frameworks/
hdiutil detach /Volumes/SDL
```

Installation SDL Image

Pour SDL Image, le paquet correspondant est [SDL_image-1.2.10.dmg](#). À installer via les commandes suivantes :

```
hdiutil attach SDL_image-1.2.10.dmg
sudo cp -r /Volumes/SDL_image/SDL_image.framework
→ /Library/Frameworks/
hdiutil detach /Volumes/SDL_image
```

Installation SDL Mixer

Pour SDL Mixer, le paquet correspondant est [SDL_mixer-1.2.12.dmg](#). À installer via les commandes suivantes :

```
hdiutil attach SDL_mixer-1.2.12.dmg
sudo cp -r /Volumes/SDL_mixer/SDL_mixer.framework
→ /Library/Frameworks/
hdiutil detach /Volumes/SDL_mixer
```

Installation SDL TTF

Pour SDL TTF, le paquet correspondant est [SDL2_ttf-2.0.12.dmg](#). À installer via les commandes suivantes :

```
hdiutil attach SDL2_ttf-2.0.12.dmg
sudo cp -r /Volumes/SDL_ttf/SDL_ttf.framework /Library/Frameworks/
hdiutil detach /Volumes/SDL_ttf
```

Installation SDL gfx

Pour SDL TTF, le paquet correspondant est [SDL_gfx-2.0.25.tar.gz](#).

Vous devrez dans un premier temps modifier les lignes suivantes du fichier `configure.in` de l'archive :

```
dnl Check for SDL
AM_PATH	SDL($SDL_VERSION,
:
AC_MSG_ERROR([*** SDL version $SDL_VERSION not found!])
)
CFLAGS="$CFLAGS $SDL_CFLAGS"
# Remove dependencies from 'sdl-config' not meant for us
→ libraries:
SDL_LIBS=$(echo "$SDL_LIBS" |sed -e 's/-lmingw32//' -e
→ 's/-lSDLmain//')
LIBS="$LIBS $SDL_LIBS"
```

Pour les remplacer par les lignes suivantes :

```
SDL_CFLAGS="-I/Library/Frameworks/SDL.framework/Headers -framework
→ SDL"
SDL_LIBS="-framework SDL"
CFLAGS="$CFLAGS $SDL_CFLAGS"
LIBS="$LIBS $SDL_LIBS -framework SDL"
```

L'installation se fera ensuite par les commandes suivantes :

```
mkdir $HOME/fakeroot
cd SDL_gfx-2.0.25
autoreconf --install
./configure --prefix=$HOME/fakeroot --without-x
make
make install
```

15.3 Fenêtre

Nous avons vu précédemment un code minimaliste permettant d'afficher et maintenir une fenêtre jusqu'à ce que l'utilisateur demande sa fermeture. SDL est une bibliothèque qui permet de travailler dans une fenêtre graphique.

15.3.1 SDL Init

Pour utiliser SDL dans une application, il faut inclure l'entête de la bibliothèque `SDL.h` qui donne accès à tous les modules par défaut de SDL. Dans le code, on annoncer l'utilisation de SDL, nous appellerons `SDL_Init` avec les options choisies. Les principales options que nous pourrions utiliser sont les suivantes :

- `SDL_INIT_VIDEO` : initialise le sous-système graphique.
- `SDL_INIT_AUDIO` : initialise le sous-système audio.
- `SDL_INIT_TIMER` : initialise le sous-système de gestion du temps.
- `SDL_INIT_EVERYTHING` : initialise tous les sous-systèmes.

Nous vérifierons que l'initialisation de SDL a fonctionné à l'aide de sa valeur de retour. En cas d'erreur le code d'erreur sera -1 et 0 en cas de succès. Notez que SDL peut allouer des ressources et peut nécessiter de mettre à l'arrêt les sous-systèmes utilisés, pensez donc à terminer le programme proprement avec `SDL_Quit` :

```
#include <SDL/SDL.h>

int main() {
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    printf("Hello SDL\n");
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

15.3.2 SDL SetVideoMode

Il est cependant bien plus intéressant d'utiliser SDL avec une fenêtre graphique. Dans SDL 1.2, la gestion de l'affichage, des images, de l'écran de la fenêtre se fait avec des surfaces `SDL_Surface *`. Ces surfaces doivent être libérées

à l'aide de `SDL_FreeSurface`. La définition de l'écran de la fenêtre se fait par `SDL_SetVideoMode` et prend les paramètres suivants :

- **Largeur** de la fenêtre.
- **Hauteur** de la fenêtre.
- **Bits par pixel**. À noter que 32 bits par pixel, c'est 8 bits par canal RGBA et donc des valeurs entières entre 0 et 255.
- **Mode de création** de la fenêtre.

Les principaux modes de création de fenêtre que nous utiliserons sont les suivants :

- `SDL_HWSURFACE` : crée une surface vidéo dans la mémoire vidéo.
- `SDL_DOUBLEBUF` : double la surface vidéo (une surface est affichée et la seconde est utilisée pour travailler dessus, la surface affichée est actualisée par celle de travail par l'appel de `SDL_Flip`). nécessite `SDL_HWSURFACE`.
- `SDL_FULLSCREEN` : lance la fenêtre en plein écran.
- `SDL_RESIZABLE` : autorise l'utilisateur à changer les dimensions de la fenêtre à la volée.
- `SDL_NOFRAME` : retire le cadre de la fenêtre, ne laissant visible que la surface.

Exemple de lancement d'une fenêtre en plein écran :

```
#include <SDL/SDL.h>

int main() {
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(800, 600, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF | SDL_FULLSCREEN)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
        → SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    printf("Hello FullScreen\n");
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

Pour garder votre fenêtre active le temps d'en visualiser le résultat, vous pouvez ajouter les lignes suivantes (nous les étudierons plus en détails au moment de la gestion des événements) :

```
int active = 1;
SDL_Event event;
while(active) {
    SDL_WaitEvent(&event);
    switch(event.type) {
        case SDL_QUIT : active = 0; break;
        case SDL_KEYUP : active = 0; break;
        default : break;
    }
}
```

Ceci permet à l'utilisateur de terminer le programme en appuyant sur une touche ou en cliquant sur la croix en haut.

16 Affichage

Pour la suite, afin de ne pas alourdir le code proposé, vous pouvez considérer que nous utiliserons le code suivant pour envelopper notre travail :

```
#include <SDL/SDL.h>

int main() {
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(800, 600, 32, SDL_HWSURFACE |
→     SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
→         SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption("Initiation à la SDL !", NULL);

    /* Votre travail ici */

    int active = 1;
    SDL_Event event;
    while(active) {
        SDL_WaitEvent(&event);
        switch(event.type) {
            case SDL_QUIT : active = 0; break;
            case SDL_KEYUP : active = 0; break;
            default : break;
        }
    }
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

16.1 Dessiner un rectangle

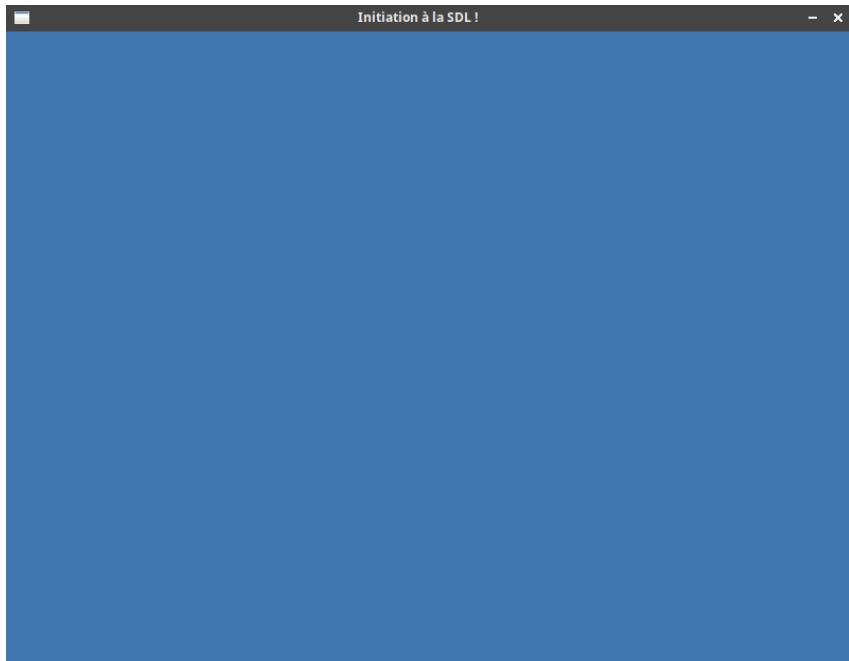
16.1.1 SDL FillRect

Pour changer la couleur de l'écran, nous pouvons appeler `SDL_FillRect`. Cette fonction colorise un rectangle de couleur unie sur une surface et prend les paramètres suivants :

- **Surface** sur laquelle le rectangle sera dessiné.
- **Rectangle** : positionnement et dimensions.
- **Couleur** : entier modélisant la couleur sur 32 bits.

Cette fonction peut par exemple être utilisée pour changer la couleur de l'écran :

```
SDL_FillRect(ecran, NULL, 0x4076ad);  
SDL_Flip(ecran);
```



4076ad est de l'hexadécimal qui peut se décomposer comme :

- 0x40 = 64 : taux de rouge sur 255.
- 0x76 = 118 : taux de vert sur 255.
- 0xad = 173 : taux de bleu sur 255.

Un logiciel avec édition de couleurs permet d'avoir ces informations. Notons que selon l'endianess de la machine, le codage de ce code couleur peut varier. Nous utiliserons donc en général la fonction `SDL_MapRGB`. Cette fonction prend en paramètres le format de pixel de la surface utilisée (informations relatives à la couleur) puis les codes couleurs des différents canaux. Le code précédent revient donc à la transformation suivante :

```
SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 64, 118, 173));
SDL_Flip(ecran);
```

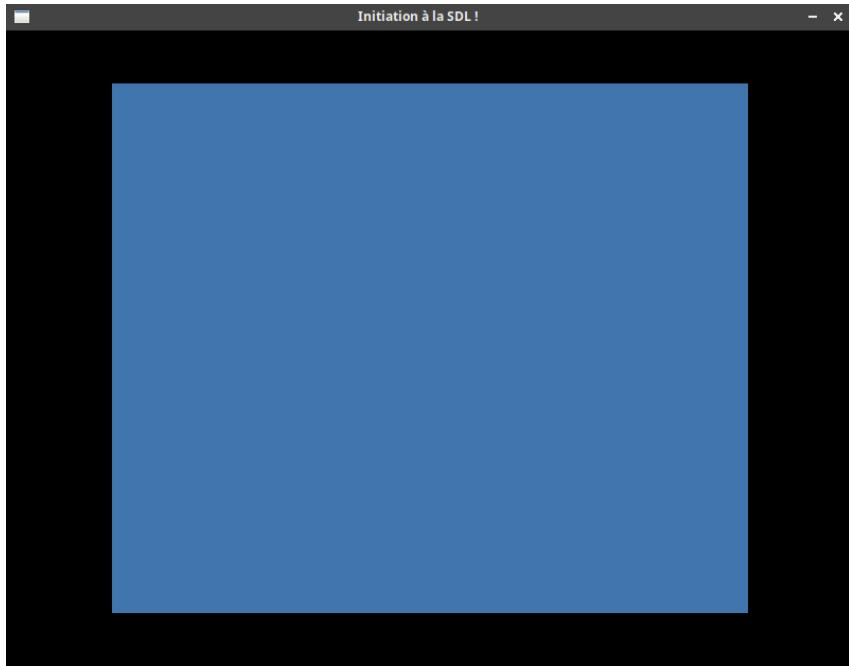
16.1.2 SDL Rect

Par défaut le paramètre du rectangle utilisé précédemment est `NULL`. Ceci indique ici que le rectangle est l'intégralité de la surface. Cependant, il est possible de le paramétriser avec une structure `SDL_Rect` dont les champs sont les suivants :

- `w` : largeur du rectangle.
- `h` : hauteur du rectangle.
- `x` : abscisse du rectangle.
- `y` : ordonnée du rectangle.

Exemple d'utilisation de `SDL_Rect` pour afficher un rectangle :

```
SDL_Rect rectangle;
rectangle.x = 100;
rectangle.y = 50;
rectangle.w = 600;
rectangle.h = 500;
SDL_FillRect(ecran, &rectangle, SDL_MapRGB(ecran->format, 64, 118, 173));
SDL_Flip(ecran);
```



16.2 Créer une surface

16.2.1 SDL_CreateRGBSurface

Jusqu'ici, la seule surface que nous ayons vue est l'écran. Cependant, il est possible de créer d'autres surfaces et les utiliser pour notre affichage. Pour créer une surface, nous utiliserons la fonction `SDL_CreateRGBSurface`. Cette fonction prend les paramètres suivants :

- **Mode de la texture :**
 - `SDL_HWSURFACE` pour définition de la surface dans la mémoire vidéo.
 - `SDL_SRCCOLORKEY` pour indiquer la présence du couleur devant être remplacée par de la transparence.
 - `SDL_SRCALPHA` pour créer un canal alpha et gérer la transparence générale.
- **Largeur.**
- **Hauteur.**
- **Bits par pixel.**
- **Masques** des canaux RGBA : rouge, vert, bleu et transparence.

Les masques des canaux de couleurs peuvent être initialisés par défaut à 0 ou générés selon la position des bits de poids forts et poids faibles dans la machine :

```
#if SDL_BYTEORDER == SDL_BIG_ENDIAN
#define RMASK 0xff000000
#define GMASK 0x00ff0000
#define BMASK 0x0000ff00
#define AMASK 0x000000ff
#define RSHIFT 24
#define GSHIFT 16
#define BSHIFT 8
#define ASHIFT 0
#else
#define RMASK 0x000000ff
#define GMASK 0x0000ff00
#define BMASK 0x00ff0000
#define AMASK 0xff000000
#define RSHIFT 0
#define GSHIFT 8
#define BSHIFT 16
#define ASHIFT 24
#endif
```

Il est donc possible de créer une surface avec `SDL_CreateRGBSurface` par le code suivant :

```
/* creation de la surface */
SDL_Surface * surface = NULL;
if((surface = SDL_CreateRGBSurface(SDL_HWSURFACE, 400, 200, 32,
→ RMASK, GMASK, BMASK, AMASK)) == NULL) {
    fprintf(stderr, "Error in SDL_CreateRGBSurface : %s\n",
    → SDL_GetError());
    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_FAILURE);
}
/* utilisation de la surface */

/* fin d'utilisation : */
SDL_FreeSurface(surface);
surface = NULL;
```

16.2.2 SDL BlitSurface

Lorsque l'on souhaite maintenant afficher une surface, nous pouvons utiliser `SDL_BlitSurface`. Cette fonction prend les paramètres suivants :

- **Surface source à afficher** : `SDL_Surface * source`.
- **Portion de la surface à afficher** : `SDL_Rect` (permet de sélectionner un morceau de l'image source et non toute la surface).
- **Surface de destination** : `SDL_Surface * destination`.
- **Position sur la surface de destination** : `SDL_Rect` (seuls les abscisses et ordonnées comptent).

Une utilisation de la surface précédente illustrant un appel à cette fonction pourrait être la suivante :

```
/* utilisation de la surface */
SDL_Rect position;
position.x = (ecran->w - surface->w) / 2;
position.y = (ecran->h - surface->h) / 2;
SDL_FillRect(surface, NULL, SDL_MapRGB(surface->format, 188, 169, 101));
SDL_BlitSurface(surface, NULL, ecran, &position);
SDL_Flip(ecran);
```



Ce type de surfaces peut aussi être utilisée pour pré-calculer une image créée procéduralement puis l'afficher (en plusieurs exemplaires) :

```

/* creation de la surface */
SDL_Surface * surface = NULL;
if((surface = SDL_CreateRGBSurface(SDL_HWSURFACE, 200, 200, 32,
→ RMASK, GMASK, BMASK, AMASK)) == NULL) {
    fprintf(stderr, "Error in SDL_CreateRGBSurface : %s\n",
    → SDL_GetError());
    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_FAILURE);
}

/* generation de l'image procedurale */
SDL_Rect rectangle;
int i, j;
float shape = 0.5; /* 0.5 : étoile, 1 : carré, 2 : disque, 8 :
→ carré arrondi */
float posNorm, radiusNorm;
radiusNorm = pow(abs(surface->w / 2), shape);
for(i = 0; i < surface->w; ++i) {
    for(j = 0; j < surface->h; ++j) {
        rectangle.x = i; rectangle.y = j;
        rectangle.w = 1; rectangle.h = 1;
        posNorm = pow(abs(i - surface->w / 2), shape) + pow(abs(j -
        → surface->h / 2), shape);
        if(posNorm < radiusNorm) {
            int val = 255 * (1. - posNorm / radiusNorm);
            SDL_FillRect(surface, &rectangle,
            → SDL_MapRGB(surface->format, val, val, val));
        }
    }
}

/* affichage de l'image */
SDL_Rect position;
position.x = (ecran->w - surface->w) / 2;
position.y = (ecran->h - surface->h) / 2;
SDL_BlitSurface(surface, NULL, ecran, &position);
position.x = (ecran->w - surface->w) - 10;
position.y = (ecran->h - surface->h) - 10;

```

```
SDL_BlitSurface(surface, NULL, ecran, &position);
position.x = 10;
position.y = 10;
SDL_BlitSurface(surface, NULL, ecran, &position);
SDL_Flip(ecran);

/* liberation */
SDL_FreeSurface(surface);
surface = NULL;
```



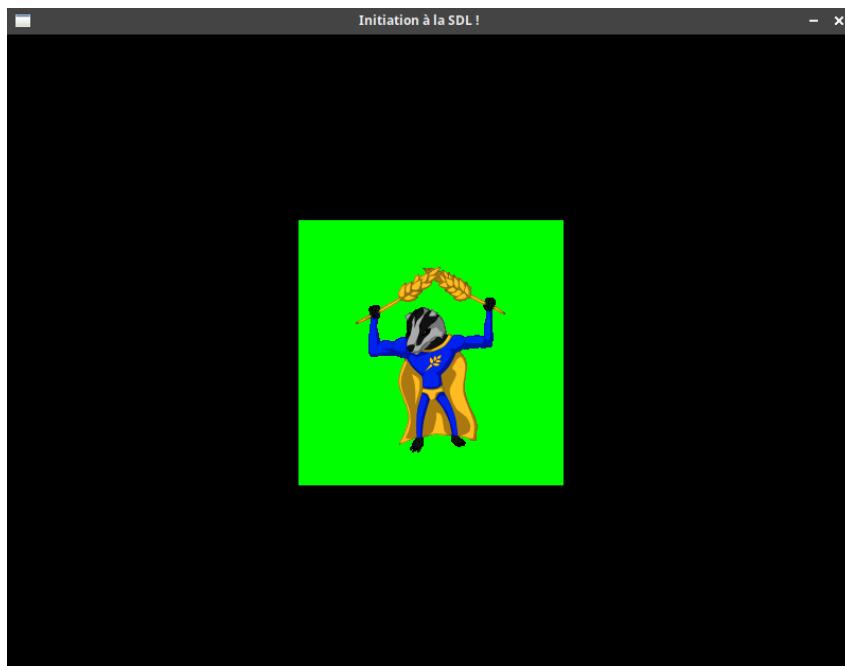
16.3 Images en BMP

Nous ne sommes pas obligés de générer des images procéduralement, il est aussi possible de charger des images matricielles depuis un fichier. Par défaut, SDL implémente le chargement d'images au format BMP. Celles-ci se chargent à l'aide de la fonction `SDL_LoadBMP` qui prend le chemin d'un fichier .bmp et renvoie une `SDL_Surface *` :

```
SDL_Surface * surface = NULL;
if((surface = SDL_LoadBMP("media/ble_heros.bmp")) == NULL) {
    fprintf(stderr, "Error in SDL_LoadBMP(\"media/ble_heros.bmp\") :
    ↵ %s\n", SDL_GetError());
    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_FAILURE);
}

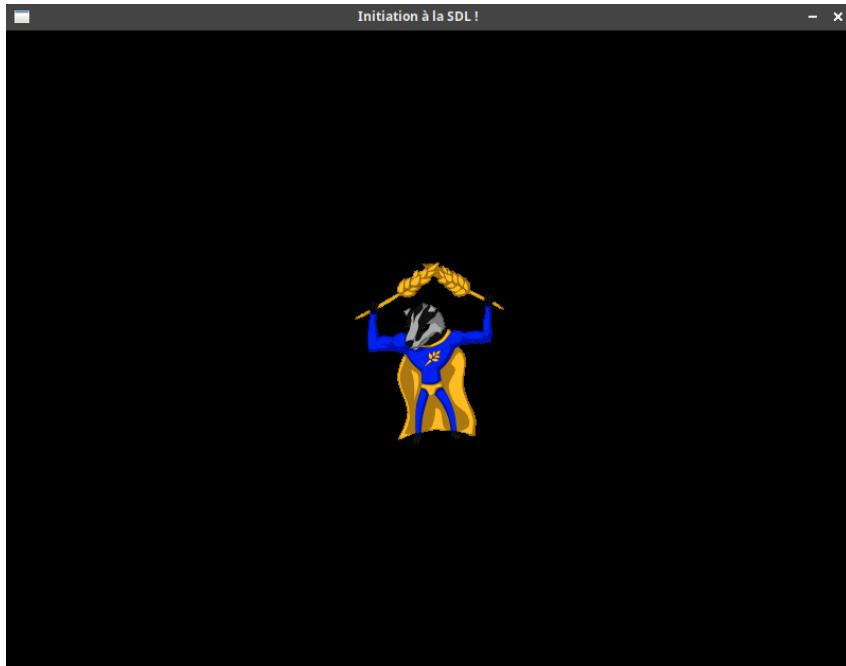
SDL_Rect position;
position.x = (ecran->w - surface->w) / 2;
position.y = (ecran->h - surface->h) / 2;
SDL_BlitSurface(surface, NULL, ecran, &position);
SDL_Flip(ecran);

SDL_FreeSurface(surface);
surface = NULL;
```



Pour ces images au format BMP, SDL propose de gérer la transparence en définissant une couleur qui peut devenir transparente à l'aide de la fonction `SDL_SetColorKey` :

```
SDL_SetColorKey(surface, SDL_SRCCOLORKEY,  
→   SDL_MapRGB(surface->format, 0, 255, 0));
```



Pour mieux visualiser la transparence de l'image, nous pouvons utiliser le fond suivant :

```
SDL_Rect rectangle;  
SDL_Color first = {32, 173, 32};  
SDL_Color second = {58, 118, 58};  
SDL_Color fond = {35, 183, 225};  
SDL_Color current;  
  
SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, fond.r,  
→   fond.g, fond.b));  
int i;  
float t;  
for(i = ecran->h / 2; i < ecran->h; ++i) {  
    t = (float)(i - ecran->h / 2) / (ecran->h - ecran->h / 2);  
    current.r = (1 - t) * first.r + t * second.r;
```

```
current.g = (1 - t) * first.g + t * second.g;
current.b = (1 - t) * first.b + t * second.b;
rectangle.x = 0; rectangle.y = i;
rectangle.w = ecran->w; rectangle.h = 1;
SDL_FillRect(ecran, &rectangle, SDL_MapRGB(ecran->format,
    → current.r, current.g, current.b));
}
```



16.3.1 SDL Image

Il est possible d'enrichir les fonctionnalités de chargement de SDL avec le module `SDL_image`. Pour ceci, les modifications à apporter pour la compilation sont les suivantes :

- Ajout du fichier d'entête `SDL/SDL_image.h` :

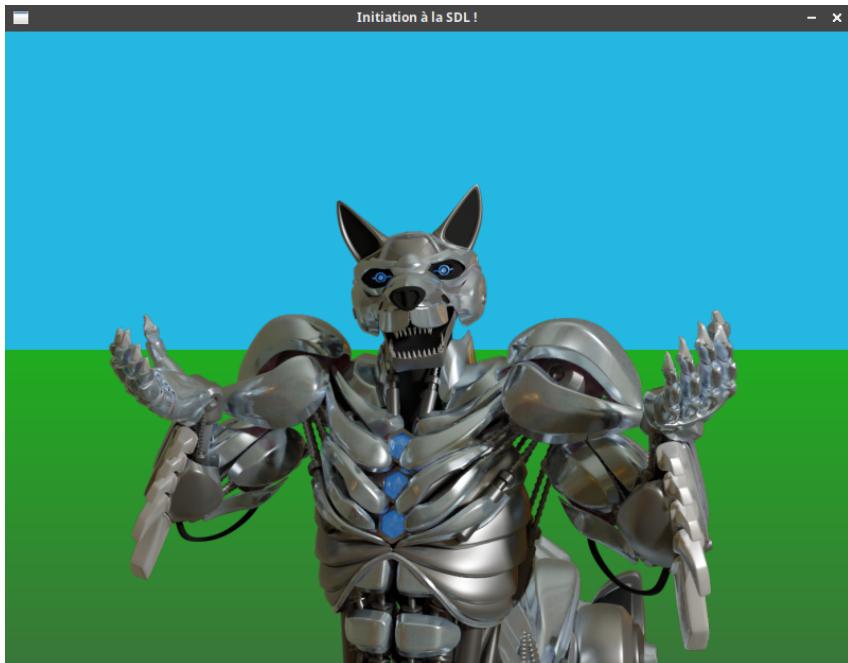
```
#include <SDL/SDL_image.h>
```

- Ajout de la bibliothèque `-lSDL_image` dans la commande de compilation :

```
gcc -o prog main.c -lSDL -lm -lSDL_image
```

Ceci permet par suite l'utilisation de `IMG_Load` prenant le chemin du fichier image en argument et renvoyant une surface :

```
SDL_Surface * surface = NULL;
if((surface = IMG_Load("media/TK.png")) == NULL) {
    fprintf(stderr, "Error in SDL_LoadBMP(\"media/TK.png\") : %s\n",
            SDL_GetError());
    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_FAILURE);
}
```

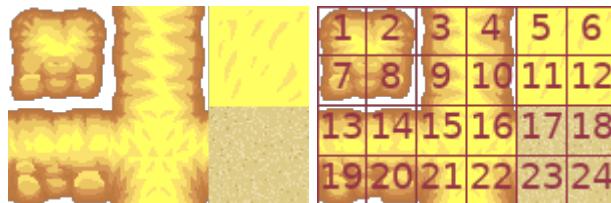


16.3.2 Dessiner une sous-image

Une astuce utilisée pour des décors peut être le tilemapping. Ceci consiste à avoir les sous-images qui constitueront un décor en une image puis extraire les

différents morceaux de sous-images pour dessin un décors dans un monde en pixel art par exemple.

Nous pouvons par exemple prendre le tileset suivant :



Pour gérer la découpe sur le tileset, nous pourrons utiliser le `SDL_Rect` proposé par `SDL_BlitSurface` pour la surface source :

```
char grille[6][8] = {
    { 1, 14, 13, 14, 13, 2, 0, 0},
    { 9, 22, 19, 20, 21, 10, 0, 0},
    { 9, 4, 1, 2, 3, 4, 0, 0},
    { 9, 16, 15, 4, 9, 16, 2, 0},
    { 7, 21, 5, 16, 15, 5, 10, 0},
    { 0, 7, 19, 20, 19, 20, 8, 0}
};

SDL_Rect tilePos;
SDL_Rect screenPos;
int i, j;
for(j = 0; j < 6; ++j) {
    for(i = 0; i < 8; ++i) {
        if(grille[j][i] == 0) continue;
        tilePos.x = 25 * ((grille[j][i] - 1) % 6);
        tilePos.y = 25 * ((grille[j][i] - 1) / 6);
        tilePos.w = 25;
        tilePos.h = 25;
        screenPos.x = i * 25;
        screenPos.y = j * 25;
        SDL_BlitSurface(surface, &tilePos, ecran, &screenPos);
    }
}
SDL_Flip(ecran);
```



16.4 Résumé

Inclusion de l'entête de la bibliothèque SDL :

```
#include <SDL/SDL.h>
```

Initialisation et arrêt de SDL :

```
SDL_Init(SDL_INIT_EVERYTHING);  
SDL_Quit();
```

Démarrage d'une fenêtre graphique :

```
/* Surface vidéo : */  
SDL_Surface * ecran = NULL;  
/* Lancement de la fenêtre graphique : */  
ecran = SDL_SetVideoMode(LARGEUR, HAUTEUR, 32, FLAGS);  
/* Rafraîchissement de l'écran */  
SDL_Flip(ecran);  
/* Libération de la fenêtre graphique : */  
SDL_FreeSurface(ecran);
```

Dessiner un rectangle / appliquer une couleur uniforme :

```
SDL_FillRect(SDL_Surface * surface, SDL_Rect * rectangle,  
→   SDL_MapRGB(surface->format, couleur.rouge, couleur.vert,  
→   couleur.bleu));
```

Création d'une surface :

```
SDL_Surface * SDL_CreateRGBSurface(FLAGS, LARGEUR, HAUTEUR, 32,  
→   RMASK, GMASK, BMASK, AMASK);
```

Impression d'une surface sur une autre :

```
SDL_BlitSurface(SDL_Surface * source, SDL_Rect * sourceRectangle,  
→   SDL_Surface * destination, SDL_Rect * destinationPosition);
```

Chargement d'une image BMP (par défaut avec SDL) et définition d'une couleur de transparence :

```
SDL_Surface * SDL_LoadBMP(const char * cheminFichier);
SDL_SetColorKey(SDL_Surface * surface, SDL_SRCCOLORKEY,
↪   SDL_MapRGB(surface->format, couleur.r, couleur.g, couleur.b));
```

Chargement d'une image avec `SDL_image` :

```
#include <SDL/SDL_image.h>
SDL_Surface * IMG_Load(const char * cheminFichier);
```

17 Événements

Nous avons vu précédemment quelques lignes pour maintenir une fenêtre active le temps que l'utilisateur appuie sur une touche ou ferme la fenêtre. Les événements avec SDL sont gérés par un type `SDL_Event`. Une proposition pour la gestion des événements est la suivante :

```
int active = 1; /* état du processus en cours */
SDL_Event event;
while(active) {
    /* récupération et traitement des événements */
    /* mise à jour de l'affichage */
}
```

17.1 Récupération d'événements

Nous avons principalement deux moyens de récupérer des événements :

- `SDL_WaitEvent` attente bloquante d'un événement.
- `SDL_PollEvent` récupération non bloquante d'événements.

17.1.1 `SDL WaitEvent`

Avec `SDL_WaitEvent`, l'attente sera bloquant et nous ne savons pas combien de temps il faudra attendre l'événement suivant. Ceci fait que pour chaque événement, nous le traitons puis nous actualisons l'affichage en conséquence.

```
int active = 1; /* état du processus en cours */
SDL_Event event;
while(active) {
    /* récupération d'un événement : */
    SDL_WaitEvent(&event);
    /* traitement de l'événement */
    /* mise à jour de l'affichage */
}
```

17.1.2 SDL_PollEvent

Dans le cas de `SDL_PollEvent` le traitement serait différent. Nous bouclerons sur les événements de la frame courante pour les traiter puis nous effectuerons l'actualisation de l'affichage. À noter qu'utiliser `SDL_PollEvent` est non-bloquant et donc ceci utiliserait votre CPU inutilement. Une proposition est d'utiliser `SDL_GetTicks` pour récupérer le temps en seconde écoulé depuis le lancement du programme et `SDL_Delay` pour mettre en attente la prochaine vérification des événements si le traitement de ceux-ci a été terminé plus vite qu'un temps réglé. On peut régler ce temps de manière à essayer de conserver un nombre d'image par secondes donné :

```
int active = 1; /* état du processus en cours */
int delay; /* gestion du temps de rafraîchissement */
const int FPS = 60; /* nombre d'images par seconde souhaité */
SDL_Event event;
while(active) {
    delay = SDL_GetTicks(); /* récupération du temps */
    /* récupération d'un événement : */
    while(SDL_PollEvent(&event)) {
        /* traitement de l'événement */
    }
    /* mise à jour de l'affichage */
    /* calcul d'un temps avec fin de boucle si besoin : */
    delay = (1000. / FPS) - (SDL_GetTicks() - delay);
    /* attente si nécessaire : */
    if(delay > 0) {
        SDL_Delay(delay);
    }
}
```

17.2 Analyse d'événements

17.2.1 Types d'événements

Lorsqu'un événement est récupéré, celui-ci possède un type. Les principaux différents types d'événements sont les suivants :

- `SDL_QUIT` : lorsque l'utilisateur clique sur la croix pour fermer la fenêtre.
- `SDL_KEYDOWN` : enfoncement d'une touche du clavier.
- `SDL_KEYUP` : relâchement d'une touche du clavier.

- `SDL_MOUSEBUTTONDOWN` : enfoncement d'un bouton de la souris.
- `SDL_MOUSEBUTTONUP` : relâchement d'un bouton de la souris.
- `SDL_MOUSEMOTION` : déplacement de la souris.

En général, un `switch` est une bonne idée pour gérer les événements :

```
switch(event.type) {
    case SDL_QUIT : {
        /* fin du processus */
        active = 0;
    } break;
    case SDL_KEYDOWN : {
        /* gestion événement touche enfoncee */
    } break;
    case SDL_KEYUP : {
        /* gestion événement touche relachee */
    } break;
    case SDL_MOUSEBUTTONDOWN : {
        /* gestion événement bouton enfoncé */
    } break;
    case SDL_MOUSEBUTTONUP : {
        /* gestion événement bouton relaché */
    } break;
    case SDL_MOUSEMOTION : {
        /* gestion événement deplacement souris */
    } break;
    default : break;
}
```

17.2.2 Clavier

Clés

Pour récupérer la touche du clavier associée à l'événement, nous accéderons au champ `key.keysym.sym` de l'événement. Les principales clés nommées sont les suivantes :

- `SDLK_ESCAPE` : touche 'Échap'.
- `SDLK_SPACE` : touche espace.
- `SDLK_RETURN` : touche entrée.
- `SDLK_BACKSPACE` : touche d'effacement du caractère précédent.

- `SDLK_DELETE` : touche 'Suppr'.
- `SDLK_0`, ..., `SDLK_9` touches numériques.
- `SDLK_a`, ..., `SDLK_z` touches alphabétiques.
- `SDLK_KP0`, ..., `SDLK_KP9` touches du keypad.
- `SDLK_UP`, `SDLK_DOWN`, `SDLK_LEFT` et `SDLK_RIGHT` flèches directionnelles.
- `SDLK_LSHIFT` et `SDLK_RSHIFT` shift.
- `SDLK_LCTRL` et `SDLK_RCTRL` 'Ctrl'.
- `SDLK_LALT` et `SDLK_RALT` 'Alt'.

Pour d'avantage de touches, il est possible de consulter la [documentation à ce sujet](#).

Le traitement des différentes touches peut ensuite se faire dans un `switch` imbriqué dans celui concernant l'état de pression de la touche :

```
switch(event.type) {
    case SDL_QUIT : {
        /* fin du processus */
        active = 0;
    } break;
    case SDL_KEYDOWN : {
        switch(event.key.keysym.sym) {
            case SDLK_ESCAPE : {
                active = 0;
            } break;
            /* ... */
            default : break;
        }
    } break;
    case SDL_KEYUP : {
        /* gestion événement touche relâchée */
    } break;
    default : break;
}
```

Activation de répétition de prise en compte

Il est possible d'activer la répétition d'une clé enfoncee pour un `SDL_KEYDOWN`. Ceci se fait par la fonction `SDL_EnableKeyRepeat`. Cette fonction prend en paramètres un délai pendant lequel la clé doit être activée pour répétition et un intervalle auquel elle sera à nouveau répétée. On place cette fonction avant la

boucle principale. Des valeurs par défaut sont fournies pour l'utilisation de cette fonction :

```
SDL_EnableKeyRepeat(SDL_DEFAULT_REPEAT_DELAY,  
→ SDL_DEFAULT_REPEAT_INTERVAL);
```

17.2.3 Boutons de souris

Lorsque l'événement est de type `SDL_MOUSEBUTTONDOWN` ou `SDL_MOUSEBUTTONUP`, il est possible de récupérer le bouton enfoncé sur la souris par l'accès à `event.button.button`. Les principales valeurs sont les suivantes :

- `SDL_BUTTON_LEFT` : clic gauche.
- `SDL_BUTTON_RIGHT` : clic droit.
- `SDL_BUTTON_MIDDLE` : bouton du milieu.
- `SDL_BUTTON_WHEELUP` : molette de la souris vers l'avant.
- `SDL_BUTTON_WHEELDOWN` : molette de la souris vers l'arrière.

17.2.4 Coordonnées de souris

Il est ensuite possible de récupérer les coordonnées de la souris lors d'une événement de type `SDL_MOUSEBUTTONDOWN` ou `SDL_MOUSEBUTTONUP` par les appels à `event.button.x` et `event.button.y` :

```
switch(event.type) {  
    case SDL_QUIT : active = 0; break;  
    case SDL_MOUSEBUTTONDOWN : {  
        switch(event.button.button) {  
            case SDL_BUTTON_LEFT : {  
                printf("clic gauche : (%d, %d)\n", event.button.x,  
                    → event.button.y);  
            } break;  
            default : break;  
        }  
    } break;  
    default : break;  
}
```

À noter que par exemple dans le cas d'un clic qui aurait été enfoncé pour déplacer un élément à l'écran, nous ne pouvons plus gérer la position de la souris dans `SDL_MOUSEBUTTONDOWN`. La souris peut être gérée lorsque déplacée par un

événement de type `SDL_MOUSEMOTION`. Pour récupérer les coordonnées de la souris, ceci demandera d'accéder aux champs `event.motion.x` et `event.motion.y`.

```
int active = 1;
SDL_Event event;
int grab = 0;
int gx, gy;
int update = 0;
while(active) {
    SDL_WaitEvent(&event);
    switch(event.type) {
        case SDL_QUIT : active = 0; break;
        case SDL_KEYDOWN : active = 0; break;
        case SDL_MOUSEBUTTONDOWN : {
            if(event.button.button == SDL_BUTTON_LEFT) {
                gx = event.button.x;
                gy = event.button.y;
                grab = 1;
                update = 1;
            }
        } break;
        case SDL_MOUSEBUTTONUP : {
            if(event.button.button == SDL_BUTTON_LEFT) {
                grab = 0;
                update = 1;
            }
        } break;
        case SDL_MOUSEMOTION : {
            if(grab) {
                gx = event.motion.x;
                gy = event.motion.y;
                update = 1;
            }
        } break;
        default : break;
    }
    if(update && grab) {
        printf("mouse grab active : (%d, %d)\n", gx, gy);
    }
    update = 0;
}
```

Imaginons que vous souhaitez que votre souris fasse tourner une caméra dans un FPS. Il est possible de récupérer le déplacement relatif de la souris par `event.motion.xrel` et `event.motion.yrel`. Vous pouvez ensuite replacer la souris au centre de l'écran avec `SDL_WarpMouse` :

```
int active = 1;
SDL_Event event;
int gx, gy;
int update = 0;
while(active) {
    SDL_WaitEvent(&event);
    switch(event.type) {
        case SDL_QUIT : active = 0; break;
        case SDL_KEYDOWN : active = 0; break;
        case SDL_MOUSEMOTION : {
            gx = event.motion.xrel;
            gy = event.motion.yrel;
            SDL_WarpMouse(ecran->w / 2, ecran->h / 2);
            update = 1;
        } break;
        default : break;
    }
    if(update) {
        printf("mouse moving : (%d, %d)\n", gx, gy);
    }
    update = 0;
}
```

17.3 Résumé

Récupération d'événements :

```
/* variable événement : */
SDL_Event event;
/* récupération bloquante : */
SDL_WaitEvent(&event);
/* récupération non bloquante : */
SDL_PollEvent(&event);
```

Analyse d'événements :

```
/* type de l'événement : */
event.type
/* touche clavier concernée : */
event.key.keysym.sym
/* bouton souris concerné : */
event.button.button
/* coordonnées position souris */
event.button.x
event.button.y
/* coordonnées position souris */
event.motion.x
event.motion.y
/* déplacement relatif souris */
event.motion.xrel
event.motion.yrel
```

Types d'événements :

```
SDL_QUIT /* croix fenêtre */
SDL_KEYDOWN /* enfoncement d'une touche du clavier */
SDL_KEYUP /* relâchement d'une touche du clavier */
SDL_MOUSEBUTTONDOWN /* enfoncement d'un bouton de la souris */
SDL_MOUSEBUTTONUP /* relâchement d'un bouton de la souris */
SDL_MOUSEMOTION /* déplacement de la souris */
```

Pour plus de détails sur les clés du clavier, se référer à la [documentation](#).

Boutons de la souris :

```
SDL_BUTTON_LEFT /* clic gauche */
SDL_BUTTON_RIGHT /* clic droit */
SDL_BUTTON_MIDDLE /* bouton du milieu */
SDL_BUTTON_WHEELUP /* molette vers l'avant */
SDL_BUTTON_WHEELDOWN /* molette vers l'arrière */
```

18 Améliorations graphiques : SDL GFX

Le module de dessin de SDL restant très limité, une extension est `SDL_GFX`. `SDL_GFX` permet dessiner d'avantage : notamment à l'aide de primitives géométriques, manipuler des images : appliquer des transformations du plan et filtrage d'images (vous pourriez explorer les fonctionnalités de `SDL_imageFilter.h` : vous pouvez passer les champs `pixels` de vos `SDL_Surface` ou jouer avec un tableau selon votre cas d'usage si pertinent).

Pour utiliser `SDL GFX`, il faudra ajouter la bibliothèque `-lSDL_gfx` dans la commande linkage et include les entêtes relatives à `SDL GFX` :

- `SDL/SDL_gfxPrimitives.h` : pour les fonctions de dessin de primitives.
- `SDL/SDL_rotozoom.h` : pour les fonctions de transformation d'une image.
- `SDL/SDL_imageFilter.h` : des fonctionnalités de traitement d'image simples.

18.1 Dessiner des primitives

Les fonctionnalités de dessin de `SDL GFX` sont données sous deux déclinaisons selon leur suffixe :

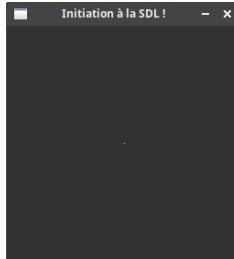
- `Color` : attend une couleur compactée sur un entier 4 octet au format `0xRRGGBBAA`.
- `RGBA` : attend 4 paramètres (teneur en rouge, vert, bleu et opacité).

18.1.1 pixel

`pixelColor` et `pixelRGBA` dessinent un pixel sur une surface aux coordonnées données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse où placer le pixel.
- `Sint16 y` : ordonnée où placer le pixel.
- `color` : couleur (`Color` ou `RGBA`).

```
pixelRGBA(ecran, ecran->w / 2, ecran->h / 2, 200, 200, 200, 255);
```



18.1.2 hline

`hlineColor` et `hlineRGBA` dessinent une ligne horizontale.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x1` : abscisse de début de la ligne.
- `Sint16 x2` : abscisse de fin de la ligne.
- `Sint16 y` : ordonnée où placer la ligne.
- `color` : couleur (`Color` ou `RGBA`).

```
hlineRGBA(ecran, ecran->w / 4, 3 * ecran->w / 4, ecran->h / 2,  
↪ 200, 200, 200, 255);
```



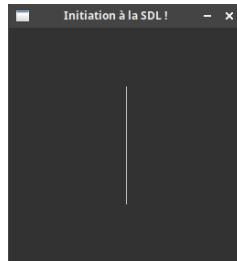
18.1.3 vline

`vlineColor` et `vlineRGBA` dessinent une ligne verticale.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse où placer la ligne.

- `Sint16 y1` : ordonnée de début de la ligne.
- `Sint16 y2` : ordonnée de fin de la ligne.
- `color` : couleur (`Color` ou `RGBA`).

```
vlineRGB(ecran, ecran->h / 2, ecran->w / 4, 3 * ecran->w / 4,
→ 200, 200, 200, 255);
```

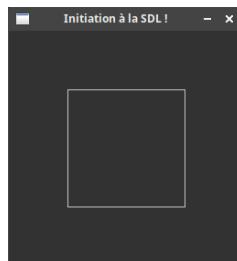


18.1.4 rectangle

`rectangleColor` et `rectangleRGBA` dessinent un rectangle depuis des extrémités données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x1` : abscisse de début du rectangle.
- `Sint16 y1` : ordonnée de début du rectangle.
- `Sint16 x2` : abscisse de fin du rectangle.
- `Sint16 y2` : ordonnée de fin du rectangle.
- `color` : couleur (`Color` ou `RGBA`).

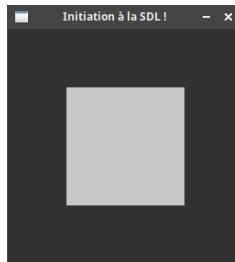
```
rectangleRGB(ecran, ecran->w / 4, ecran->w / 4, 3 * ecran->w / 4,
→ 3 * ecran->w / 4, 200, 200, 200, 255);
```



box (filled rectangle)

Version remplie du rectangle.

```
boxRGB(ecran, ecran->w / 4, ecran->w / 4, 3 * ecran->w / 4, 3 *
↪ ecran->w / 4, 200, 200, 200, 255);
```

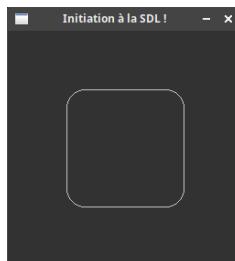


18.1.5 roundedRectangle

`roundedRectangleColor` et `roundedRectangleRGBA` dessinent un rectangle depuis des extrémités données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x1` : abscisse de début du rectangle.
- `Sint16 y1` : ordonnée de début du rectangle.
- `Sint16 x2` : abscisse de fin du rectangle.
- `Sint16 y2` : ordonnée de fin du rectangle.
- `Sint16 rad` : largeur de l'arrondi.
- `color` : couleur (`Color` ou `RGBA`).

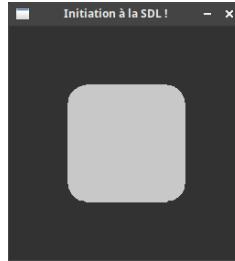
```
roundedRectangleRGBA(ecran, ecran->w / 4, ecran->w / 4, 3 *
↪ ecran->w / 4, 3 * ecran->w / 4, 20, 200, 200, 200, 255);
```



roundedBox (filled rounded rectangle)

Version remplie du rectangle arrondi.

```
roundedBoxRGBA(ecran, ecran->w / 4, ecran->w / 4, 3 * ecran->w /
→ 4, 3 * ecran->w / 4, 20, 200, 200, 200, 255);
```

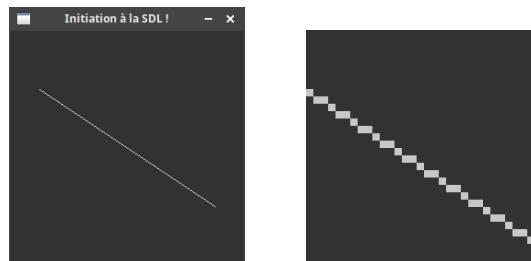


18.1.6 line

`lineColor` et `lineRGBA` dessinent une ligne depuis des extrémités données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x1` : abscisse de début de la ligne.
- `Sint16 y1` : ordonnée de début de la ligne.
- `Sint16 x2` : abscisse de fin de la ligne.
- `Sint16 y2` : ordonnée de fin de la ligne.
- `color` : couleur (`Color` ou `RGBA`).

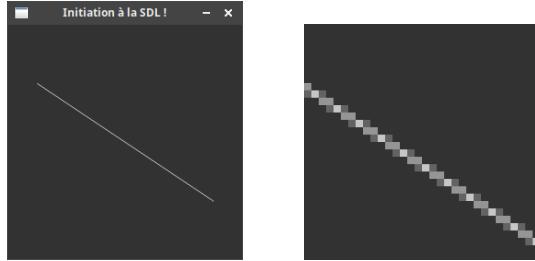
```
lineRGBA(ecran, ecran->w / 8, ecran->w / 4, 7 * ecran->w / 8, 3 *
→ ecran->w / 4, 200, 200, 200, 255);
```



aaline

Version adoucie de la ligne (avec anti-aliasing).

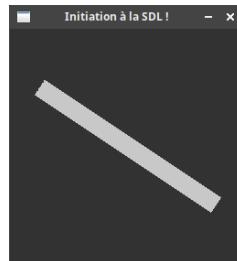
```
aalineRGBA(ecran, ecran->w / 8, ecran->w / 4, 7 * ecran->w / 8, 3
↪ * ecran->w / 4, 200, 200, 200, 255);
```



thickLine

Version d'épaisseur réglable de la ligne (prend l'épaisseur en argument supplémentaire).

```
thickLineRGBA(ecran, ecran->w / 8, ecran->w / 4, 7 * ecran->w / 8,
↪ 3 * ecran->w / 4, 20, 200, 200, 200, 255);
```



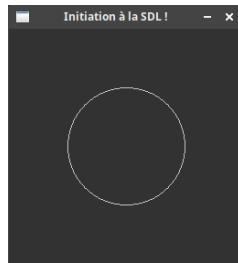
18.1.7 circle

`circleColor` et `circleRGBA` dessinent un cercle depuis un centre et un rayon donnés.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse du centre du cercle.

- `Sint16 y` : ordonnée du centre du cercle.
- `Sint16 rad` : rayon du cercle.
- `color` : couleur (Color ou RGBA).

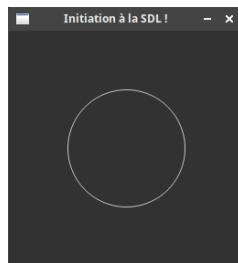
```
circleRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4, 200,
→ 200, 200, 255);
```



aacircle

Version avec anti-aliasing du cercle.

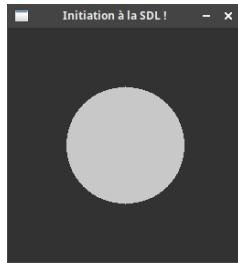
```
aacircleRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4, 200,
→ 200, 200, 255);
```



filledCircle

Version remplie du cercle.

```
filledCircleRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4,
→ 200, 200, 200, 255);
```

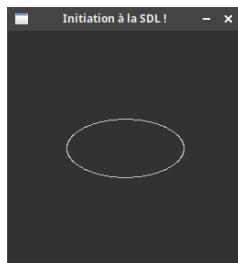


18.1.8 ellipse

`ellipseColor` et `ellipseRGBA` dessinent une ellipse depuis un centre et deux rayons (en largeur et en hauteur) donnés.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse du centre du cercle.
- `Sint16 y` : ordonnée du centre du cercle.
- `Sint16 rx` : largeur de l'ellipse.
- `Sint16 ry` : hauteur de l'ellipse.
- `color` : couleur (`Color` ou `RGBA`).

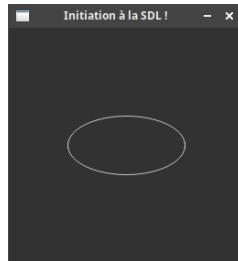
```
ellipseRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4,  
↪ ecran->w / 8, 200, 200, 200, 255);
```



aaellipse

Version avec anti-aliasing de l'ellipse.

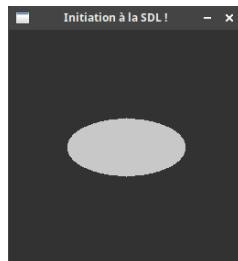
```
aaellipseRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4,  
↪ ecran->w / 8, 200, 200, 200, 255);
```



[filledEllipse](#)

Version remplie de l'ellipse.

```
filledEllipseRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4,
                  ecran->w / 8, 200, 200, 200, 255);
```

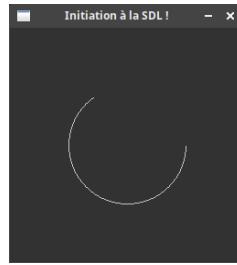


18.1.9 arc

`arcColor` et `arcRGBA` dessinent un arc de cercle depuis un centre, un rayon et des angles (en degrés) donnés.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse du centre du cercle.
- `Sint16 y` : ordonnée du centre du cercle.
- `Sint16 rad` : rayon du cercle.
- `Sint16 startAngle` : angle de début de l'arc de cercle.
- `Sint16 endAngle` : angle de fin de l'arc de cercle.
- `color` : couleur (`Color` ou `RGBA`).

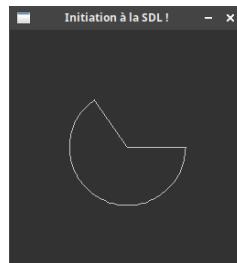
```
arcRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4, 0, 235,  
        ↳ 200, 200, 200, 255);
```



pie

Version fermée de l'arc de cercle.

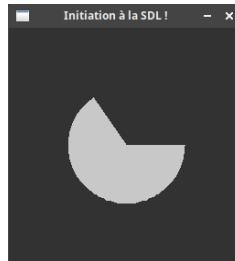
```
pieRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4, 0, 235,  
        ↳ 200, 200, 200, 255);
```



filledPie

Version fermée et remplie de l'arc de cercle.

```
filledPieRGBA(ecran, ecran->w / 2, ecran->w / 2, ecran->w / 4, 0,  
              ↳ 235, 200, 200, 200, 255);
```

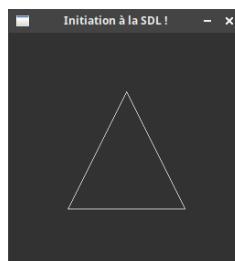


18.1.10 trigon

`trigonColor` et `trigonRGBA` dessinent un triangle depuis des sommets données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x1` : abscisse du premier sommet du triangle.
- `Sint16 y1` : ordonnée du premier sommet du triangle.
- `Sint16 x2` : abscisse du second sommet du triangle.
- `Sint16 y2` : ordonnée de second sommet du triangle.
- `Sint16 x3` : abscisse du troisième sommet du triangle.
- `Sint16 y3` : ordonnée de troisième sommet du triangle.
- `color` : couleur (Color ou RGBA).

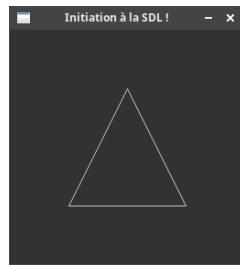
```
trigonRGBA(ecran, ecran->w / 2, ecran->w / 4, ecran->w / 4, 3 *
    ↵ ecran->w / 4, 3 * ecran->w / 4, 3 * ecran->w / 4, 200, 200,
    ↵ 200, 255);
```



aatrigon

Version avec anti-aliasing du triangle.

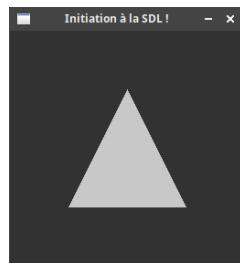
```
aatrigonRGBA(ecran, ecran->w / 2, ecran->w / 4, ecran->w / 4, 3 *
    ↵ ecran->w / 4, 3 * ecran->w / 4, 3 * ecran->w / 4, 200, 200,
    ↵ 200, 255);
```



filledTrigon

Version remplie du triangle.

```
filledTrigonRGBA(ecran, ecran->w / 2, ecran->w / 4, ecran->w / 4,
    ↵ 3 * ecran->w / 4, 3 * ecran->w / 4, 3 * ecran->w / 4, 200,
    ↵ 200, 200, 255);
```



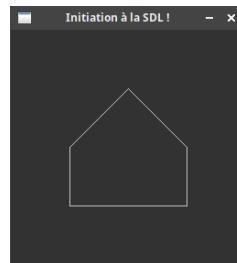
18.1.11 polygon

`polygonColor` et `polygonRGBA` dessinent un polygone depuis des sommets données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 * xs` : abscisses des sommets du polygone.
- `Sint16 * ys` : ordonnées des sommets du polygone.
- `int n` : nombre de sommets.

- `color` : couleur (Color ou RGBA).

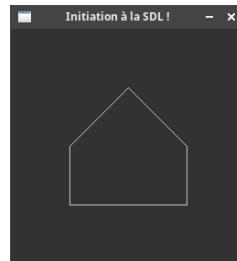
```
Sint16 xs[] = {ecran->w / 4, ecran->w / 2, 3 * ecran->w / 4, 3 *
    ↵ ecran->w / 4, ecran->w / 4};
Sint16 ys[] = {ecran->h / 2, ecran->h / 4, ecran->h / 2, 3 *
    ↵ ecran->h / 4, 3 * ecran->h / 4};
polygonRGBA(ecran, xs, ys, 5, 200, 200, 200, 255);
```



aapolygon

Version avec anti-aliasing du polygone.

```
Sint16 xs[] = {ecran->w / 4, ecran->w / 2, 3 * ecran->w / 4, 3 *
    ↵ ecran->w / 4, ecran->w / 4};
Sint16 ys[] = {ecran->h / 2, ecran->h / 4, ecran->h / 2, 3 *
    ↵ ecran->h / 4, 3 * ecran->h / 4};
aapolygonRGBA(ecran, xs, ys, 5, 200, 200, 200, 255);
```



filledPolygon

Version remplie du polygone.

```
Sint16 xs[] = {ecran->w / 4, ecran->w / 2, 3 * ecran->w / 4, 3 *
    ↵ ecran->w / 4, ecran->w / 4};
Sint16 ys[] = {ecran->h / 2, ecran->h / 4, ecran->h / 2, 3 *
    ↵ ecran->h / 4, 3 * ecran->h / 4};
filledPolygonRGBA(ecran, xs, ys, 5, 200, 200, 200, 255);
```

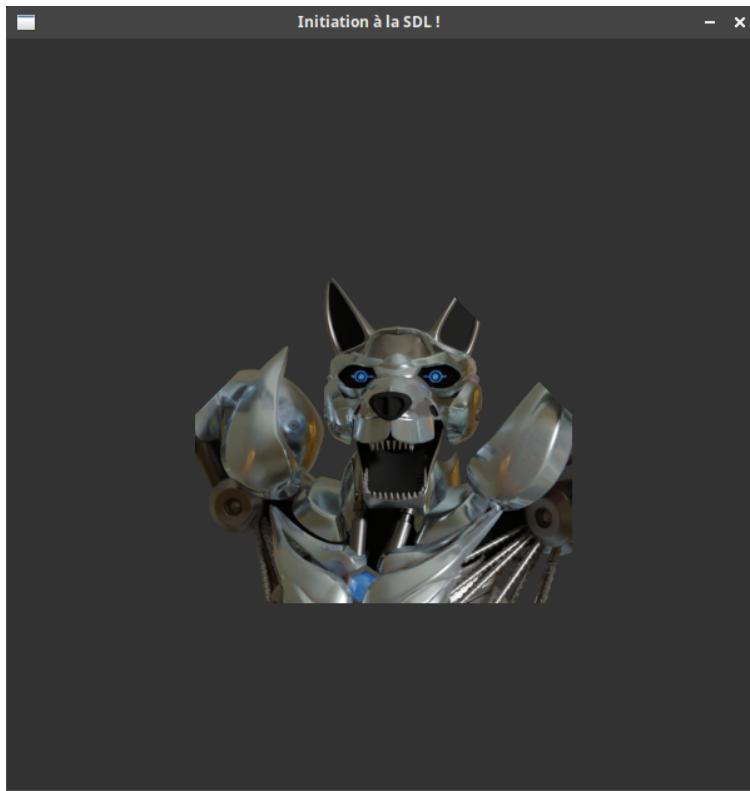


18.1.12 texturedPolygon

`texturedPolygon` dessine une surface dans la zone donnée par un polygone exprimé par des sommets données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 * xs` : abscisses des sommets du polygone.
- `Sint16 * ys` : ordonnées des sommets du polygone.
- `int n` : nombre de sommets.
- `SDL_Surface * texture` : surface à dessiner.
- `int texture_dx` : déplacement inverse en abscisse de la texture.
- `int texture_dy` : déplacement inverse en ordonnée de la texture.

```
Sint16 xs[] = {ecran->w / 4, ecran->w / 2, 3 * ecran->w / 4, 3 *
    ↵ ecran->w / 4, ecran->w / 4};
Sint16 ys[] = {ecran->h / 2, ecran->h / 4, ecran->h / 2, 3 *
    ↵ ecran->h / 4, 3 * ecran->h / 4};
texturedPolygon(ecran, xs, ys, 5, surface, 0, -ecran->h / 16);
```



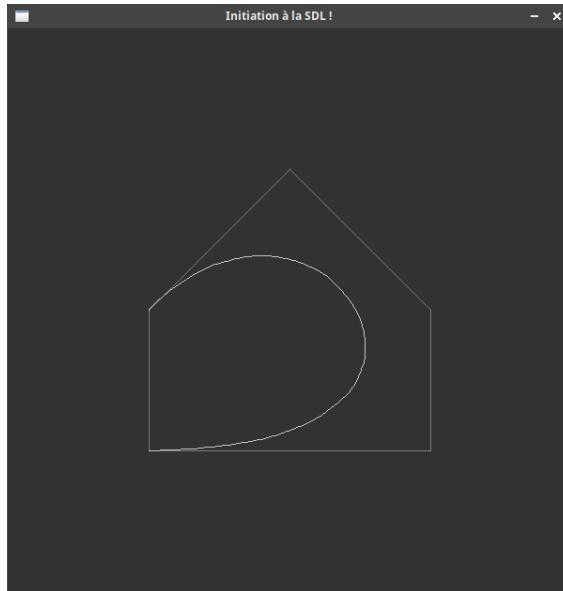
18.1.13 bezier

`bezierColor` et `bezierRGBA` dessinent une courbe de Bezier selon des sommets données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 * xs` : abscisses des sommets du polygone.
- `Sint16 * ys` : ordonnées des sommets du polygone.
- `int n` : nombre de sommets.
- `int s` : nombre d’itérations pour affiner le rendu de la courbe (au moins 2).
- `color` : couleur (`Color` ou `RGBA`).

```
Sint16 xs[] = {ecran->w / 4, ecran->w / 2, 3 * ecran->w / 4, 3 *
               ecran->w / 4, ecran->w / 4};
Sint16 ys[] = {ecran->w / 2, ecran->w / 4, ecran->w / 2, 3 *
               ecran->w / 4, 3 * ecran->w / 4};
```

```
polygonRGBA(ecran, xs, ys, 5, 200, 200, 200, 127);  
bezierRGBA(ecran, xs, ys, 5, 10, 200, 200, 200, 255);
```



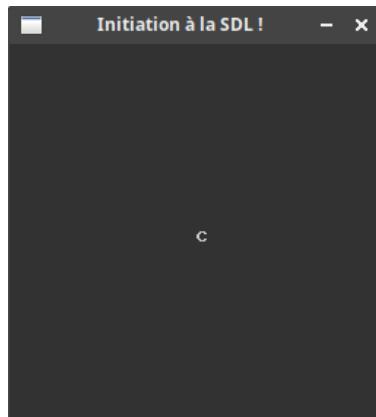
18.2 Écrire du texte

18.2.1 character

`characterColor` et `characterRGBA` dessinent un caractère sur une surface aux coordonnées données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse où placer le caractère.
- `Sint16 y` : ordonnée où placer le caractère.
- `char c` : caractère à dessiner.
- `color` : couleur (`Color` ou `RGBA`).

```
characterRGBA(ecran, ecran->w / 2, ecran->h / 2, 'C', 200, 200,  
→ 200, 255);
```

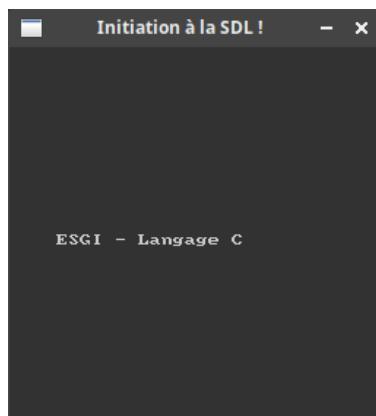


18.2.2 string

`stringColor` et `stringRGBA` dessinent une chaîne de caractères sur une surface aux coordonnées données.

- `SDL_Surface * dst` : surface où dessiner.
- `Sint16 x` : abscisse où placer le caractère.
- `Sint16 y` : ordonnée où placer le caractère.
- `const char * s` : chaîne de caractères à dessiner.
- `color` : couleur (Color ou RGBA).

```
stringRGBA(ecran, ecran->w / 8, ecran->h / 2, "ESGI - Langage C",
→ 200, 200, 200, 255);
```



18.3 Faire tourner une image

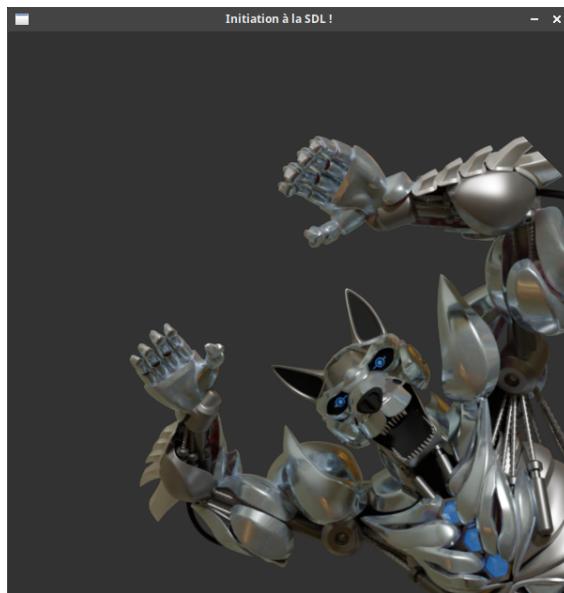
Pour certains cas d'usage, comme des personnages vus de haut qui se dirigeraient dans une direction, il peut être intéressant de réussir à faire tourner les images directement en jeu plutôt que le faire artistiquement pour tous les cas. De même nous pourrions vouloir différentes tailles pour une image : pour des personnages de taille variable ou lors d'un agrandissement / éloignement de la carte par le joueur. Ces fonctionnalités sont disponibles depuis l'import de l'en-tête `SDL/SDL_rotozoom.h`.

18.3.1 rotozoomSurface

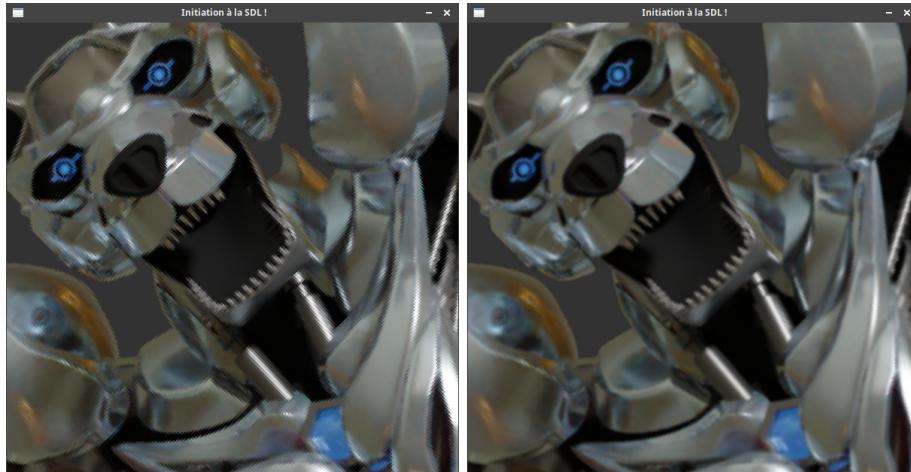
La fonction `rotozoomSurface` permet à la fois de faire tourner une surface mais aussi de changer sa rotation : elle renvoie la surface calculée depuis celle donnée et les paramètres souhaités.

- `SDL_Surface * surface` : surface de référence.
- `double angle` : angle de rotation (en degrés).
- `double zoom` : facteur d agrandissement (1 pour conserver échelle).
- `int smooth` : option de dé-pixelisation.

```
rotozoomSurface(surface, 45, 1., 0)
```



```
/* Sans adoucissement : à gauche */
rotozoomSurface(surface, 45, 3., 0)
/* Avec adoucissement : à droite */
rotozoomSurface(surface, 45, 3., 1)
```

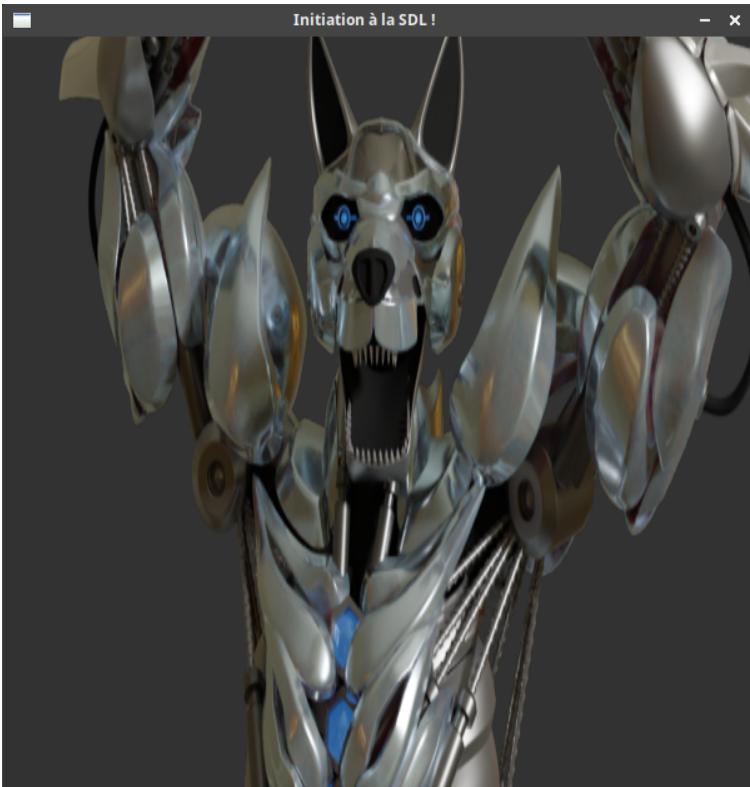


18.3.2 zoomSurface

Dans le cas où il n'y aurait pas de rotation, il est possible d'avoir un zoom avec un facteur d'agrandissement horizontal et vertical qui diffère. Pour ceci, nous pouvons utiliser la fonction `zoomSurface` :

- `SDL_Surface * surface` : surface de référence.
- `double zx` : facteur d'agrandissement en largeur.
- `double zy` : facteur d'agrandissement en hauteur.
- `int smooth` : option de dé-pixelisation.

```
zoomSurface(surface, 1., 2., 1)
```



18.4 Quelques usages de `SDL_imageFilter`

`SDL_imageFilter` offre quelques fonctionnalités de traitement d'images directement depuis ses pixels. Les pixels d'une `SDL_Surface` s'obtiennent depuis son champ `pixels`. Le défaut de ces fonctionnalités est qu'il considère une image uniquement comme une suite d'octets, là où vous pourriez vouloir améliorer ces traitements. Nous proposerons quelques illustrations des fonctionnalités sur l'association d'une référence à deux possibilités lorsqu'une autre image est demandée.

18.4.1 Références

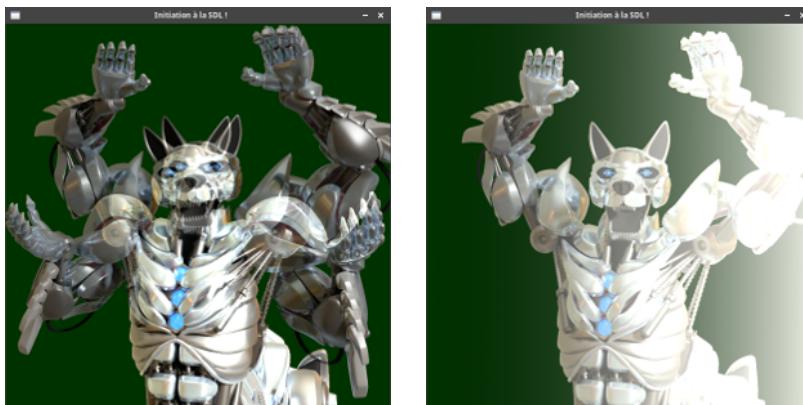
Les images utilisées pour illustrer les fonctionnalités de `SDL_imageFilter` sont les suivantes (rendues via SDL sur un fond vert) :



18.4.2 `SDL_imageFilterAdd`

Ajoute deux images octet par octet.

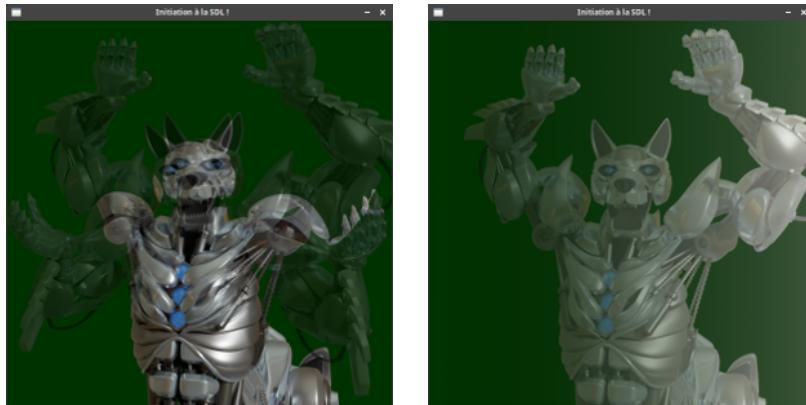
```
SDL_imageFilterAdd(surface->pixels, surface2->pixels,
                   resultat->pixels, surface->w * surface->h * 4);
```



18.4.3 `SDL_imageFilterMean`

Fait la moyenne de deux images octet par octet.

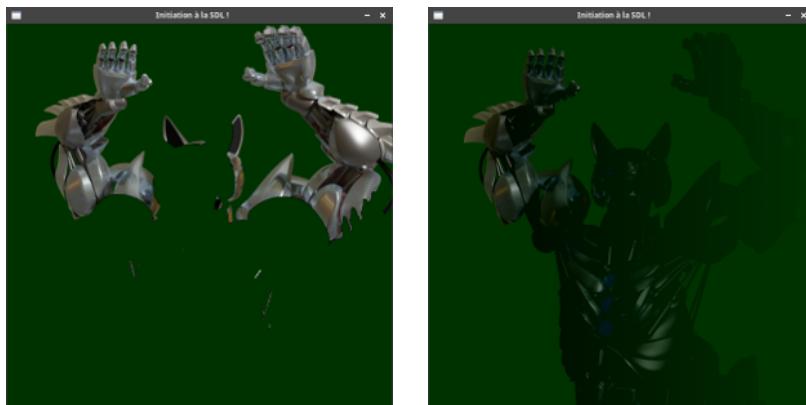
```
SDL_imageFilterMean(surface->pixels, surface2->pixels,  
                   surface3->pixels, surface->w * surface->h * 4);
```



18.4.4 `SDL_imageFilterSub`

Fait la soustraction de deux images octet par octet.

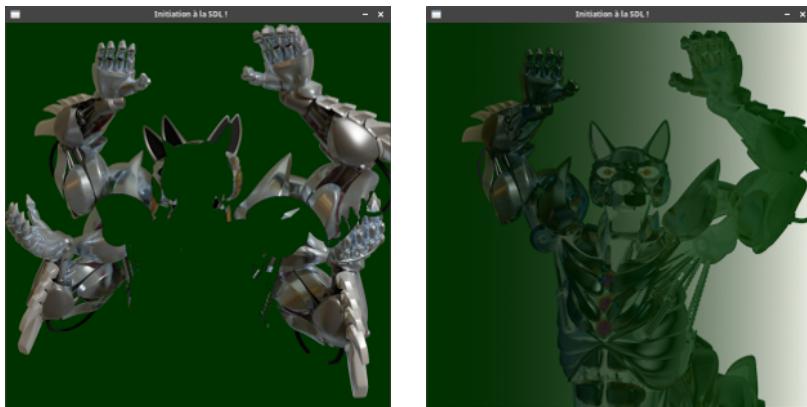
```
SDL_imageFilterSub(surface->pixels, surface2->pixels,  
                   surface3->pixels, surface->w * surface->h * 4);
```



18.4.5 `SDL_imageFilterAbsDiff`

Fait la différence symétrique de deux images octet par octet.

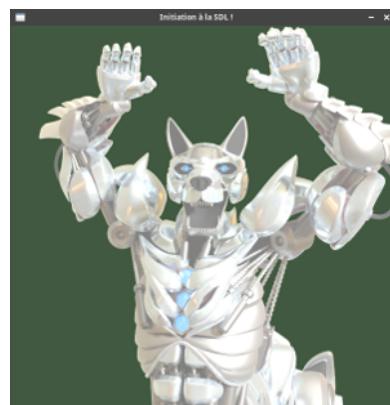
```
SDL_imageFilterAbsDiff(surface->pixels, surface2->pixels,  
→ surface3->pixels, surface->w * surface->h * 4);
```



18.4.6 `SDL_imageFilterAddByte`

Ajoute une valeur uniforme à tous les octets.

```
SDL_imageFilterAddByte(surface->pixels, surface3->pixels,  
→ surface->w * surface->h * 4, 127);
```



18.4.7 **SDL_imageFilterSubByte**

Soustrait une valeur uniforme à tous les octets.

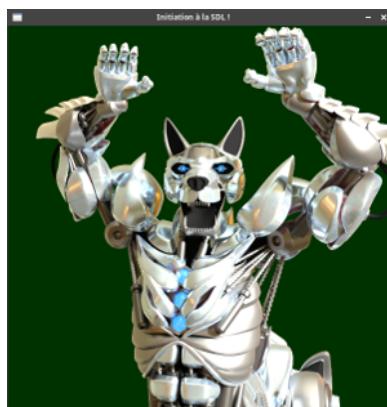
```
SDL_imageFilterSubByte(surface->pixels, surface3->pixels,  
↪ surface->w * surface->h * 4, 127);
```



18.4.8 **SDL_imageFilterMultByByte**

Multiplie une valeur uniforme à tous les octets.

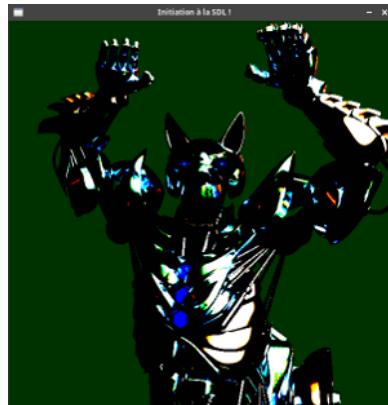
```
SDL_imageFilterMultByByte(surface->pixels, surface3->pixels,  
↪ surface->w * surface->h * 4, 2);
```



18.4.9 `SDL_imageFilterBinarizeUsingThreshold`

Maximise ou Minimise la valeur selon un seuil donné.

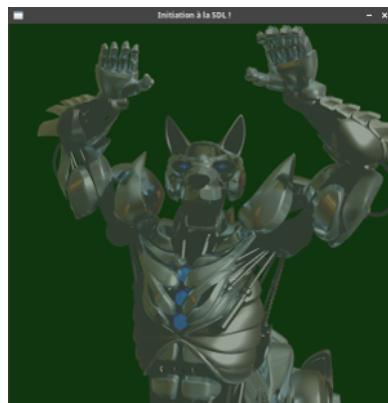
```
SDL_imageFilterBinarizeUsingThreshold(surface->pixels,  
→ surface3->pixels, surface->w * surface->h * 4, 127);
```



18.4.10 `SDL_imageFilterClipToRange`

Replace en bordure d'un intervalle les valeurs en sortant.

```
SDL_imageFilterClipToRange(surface->pixels, surface3->pixels,  
→ surface->w * surface->h * 4, 63, 196);
```



19 Texte : SDL TTF

Bien que SDL GFX propose l'écriture de texte, celle-ci reste très limitée. En particulier, nous pourrions souhaiter utiliser un font personnalisé ou différentes tailles de texte. Ceci est possible depuis la bibliothèque `SDL_TTF`.

19.1 Mise en place

Pour utiliser `SDL TTF`, il faudra ajouter la bibliothèque dans la commande de linkage par `-lSDL_ttf` et inclure son entête `SDL/SDL_ttf.h`.

Avant utilisant, il faut appeler `TTF_Init()`. De même, il faudra penser à appeler `TTF_Quit()` en fin de programme.

19.2 Ouverture d'un Font

Pour utiliser les fonctionnalités de `SDL TTF`, il faut dans un premier temps charger un `TTF_Font *`. Ceci se fait par l'appel à `TTF_OpenFont` :

- `const char * file` : chemin du fichier `.ttf` à ouvrir.
- `int ptsize` : taille de la police à utiliser.

Il faudra penser à fermer ce `TTF_Font *` à l'aide de `TTF_CloseFont`.

```
TTF_Font * police = NULL;
const char * police_path = "media/arial.ttf";
if((police = TTF_OpenFont(police_path, 40)) == NULL) {
    fprintf(stderr, "Erreur d'ouverture de la police \"%s\"\n",
            police_path);
    TTF_Quit();
    exit(EXIT_FAILURE);
}
```

19.3 Générer une image depuis un texte

Une fois un font à disposition, il est possible de l'utiliser pour générer des `SDL_Surface` * à partir de texte et celui-ci. Ceci se fait par l'appel à une fonction type `TTF_RenderText`. À noter que pour gérer des caractères accentués, il faudrait utiliser sa variante `TTF_RenderUTF8` par exemple.

Les principales manières de rendre du texte sont les suivantes :

- `Solid` : texte en transparence simple.
- `Shaded` : texte sur un fond coloré.
- `Blended` : texte en transparence avec anti-aliasing.

Les fonctions `TTF_Render[CHARSET]_[RENDEU]` prennent les arguments suivants :

- `TTF_Font * font` : font à utiliser.
- `const char * text` : texte à rendre.
- `SDL_Color color` : couleur du texte.
- `SDL_Color fond_color` : couleur du fond pour le rendu `Shaded`.

Pour afficher du texte dans les différentes méthodes de rendu, ceci pourrait se faire de la manière suivante :

```
SDL_Rect position;
const char * texte = "ESGI - Langage C : exemple de texte";

position.x = 50;
position.y = 50;

surface = TTF_RenderText_Solid(police, texte, (SDL_Color){200,
    ↵ 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
position.y = 150;
```

```
SDL_FreeSurface(surface);
surface = TTF_RenderText_Shaded(police, texte, (SDL_Color){200,
    → 200, 200}, (SDL_Color){100, 100, 100});

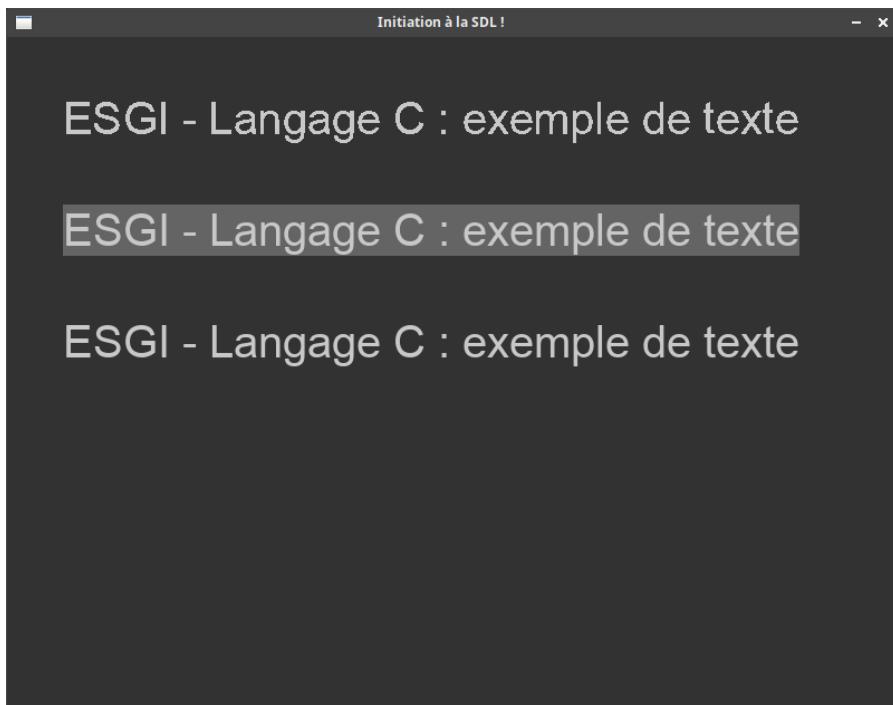
SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
position.y = 250;

SDL_FreeSurface(surface);
surface = TTF_RenderText_Blended(police, texte, (SDL_Color){200,
    → 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);
```

Et donner le rendu suivant :



19.4 Effets de style

Il est possible d'ajouter des effets de style à l'aide de la fonction `TTF_SetFontStyle`. Cette fonction prend en arguments une police et une combinaison de styles :

- `TTF_STYLE_NORMAL` : aucun style.
- `TTF_STYLE_BOLD` : texte en gras.
- `TTF_STYLE_ITALIC` : texte en italique.
- `TTF_STYLE_UNDERLINE` : texte souligné.
- `TTF_STYLE_STRIKETHROUGH` : texte barré.

À noter qu'un autre moyen de styliser le texte est `TTF_SetFontOutline` permettant de vider le texte et ne garder que son contour.

Ces effets de style peuvent s'illustrer par le code suivant :

```
SDL_Rect position;
position.x = 50;
position.y = 50;

TTF_SetFontStyle(police, TTF_STYLE_BOLD);
surface = TTF_RenderUTF8_Blended(police, "Texte en gras",
→ (SDL_Color){200, 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
position.y = 150;

TTF_SetFontStyle(police, TTF_STYLE_ITALIC);
SDL_FreeSurface(surface);
surface = TTF_RenderUTF8_Blended(police, "Texte en italique",
→ (SDL_Color){200, 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
```

```
position.y = 250;

TTF_SetFontStyle(police, TTF_STYLE_UNDERLINE);
SDL_FreeSurface(surface);
surface = TTF_RenderUTF8_Blended(police, "Texte souligné",
→ (SDL_Color){200, 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
position.y = 250;

TTF_SetFontStyle(police, TTF_STYLE_STRIKETHROUGH);
SDL_FreeSurface(surface);
surface = TTF_RenderUTF8_Blended(police, "Texte barré",
→ (SDL_Color){200, 200, 200});

position.x = 50;
position.y = 350;

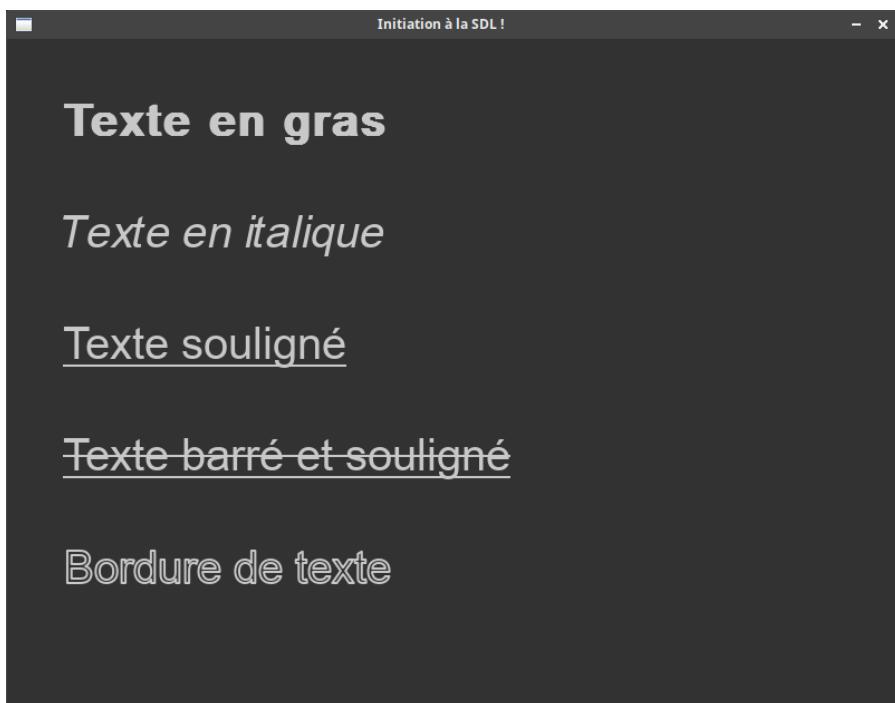
TTF_SetFontStyle(police, TTF_STYLE_UNDERLINE |
→ TTF_STYLE_STRIKETHROUGH);
SDL_FreeSurface(surface);
surface = TTF_RenderUTF8_Blended(police, "Texte barré et
→ souligné", (SDL_Color){200, 200, 200});

SDL_BlitSurface(surface, NULL, ecran, &position);

position.x = 50;
position.y = 450;

TTF_SetFontStyle(police, TTF_STYLE_NORMAL);
TTF_SetFontOutline(police, 1);
SDL_FreeSurface(surface);
```

```
surface = TTF_RenderUTF8_Blended(police, "Bordure de texte",
→ (SDL_Color){200, 200, 200});  
  
SDL_BlitSurface(surface, NULL, ecran, &position);
```



20 Musique : SDL Mixer

Votre application est maintenant visuelle et graphique, mais il manque le son, pour ceci, vous pourriez utiliser **SDL_Mixer** pour donner d'avantage vie à votre programme. **SDL_Audio** existe mais reste relativement limité et minimaliste. C'est une extension assimilable à **SDL_Image** et **SDL_GFX** pour manipuler des images en plus de la base de **SDL**.

20.1 Mise en place

L'utilisation de **SDL Mixer** nécessite l'ajout de `-lSDL_mixer` dans la commande de linkage et l'import de l'entête **SDL/SDL_mixer.h**.

Une fonction **Mix_Init** existe mais reste optionnelle : son intérêt est d'indiquer si vous souhaitez utiliser des types spécifiques : **MIX_INIT_FLAC**, **MIX_INIT_MOD**, **MIX_INIT_MP3**, **MIX_INIT_OGG** et d'autres. Ceci permet de charger les bibliothèques dynamiques pour l'ouverture des formats à l'avance. Sinon ceci est fait lorsqu'un type est rencontré. Nous terminerons par l'appel à **Mix_Quit()**. Il faudra cependant ajouter le flag **SDL_INIT_AUDIO** lors du **SDL_Init**. À noter qu'il est compris dans **SDL_INIT_EVERYTHING**.

Nous devons ensuite ouvrir un service audio pour jouer de la musique tout comme nous avons un écran pour afficher des images. Ceci se fait via **Mix_OpenAudio** (qu'il faudra fermer via **Mix_CloseAudio()**) :

- **int frequency** : fréquence en Hertz pour jouer vos sons. En général, un choix acceptable est **MIX_DEFAULT_FREQUENCY** (peut correspondre à 22050 ou 44100 selon votre version de **SDL Mix**).
- **Uint16 format** : format audio (relatif à **SDL Audio**), voir **SDL Audio** si besoin spécifique, sinon **MIX_DEFAULT_FORMAT** est un choix pertinent.
- **int channels** : choix de canaux en mono (1) ou stéréo (2).
- **int chunksizes** : varie selon l'utilisation, si vous utilisez majoritairement des sons courts, une basse taille permettra un chargement rapide des morceaux.

Si vous privilégiez des sons longs ou musique longue, une plus grande taille éviterait des latences à devoir changer trop rapidement d'un chunk à l'autre dans votre son. En général, un choix acceptable semble être 512.

Votre installation de départ peut ressembler au code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_mixer.h>

int main() {
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if(Mix_Init(MIX_INIT_MP3) != 0) {
        fprintf(stderr, "Error in Mix_Init : %s\n", Mix_GetError());
        exit(EXIT_FAILURE);
    }
    if(Mix_OpenAudio(
        MIX_DEFAULT_FREQUENCY,
        MIX_DEFAULT_FORMAT,
        MIX_DEFAULT_CHANNELS,
        MIX_DEFAULT_CHANNELS * 512) != 0) {
        fprintf(stderr, "Error in Mix_OpenAudio : %s\n",
        ↪ Mix_GetError());
        exit(EXIT_FAILURE);
    }

    /* Vos instructions */

    Mix_CloseAudio();
    Mix_Quit();
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

À noter que ceci ne demande pas forcément qu'un service de fenêtre soit ouvert pour être utilisé.

20.2 Jouer une musique

20.2.1 Chargement d'une musique

Le service que vous avez ouvert vous permet maintenant de jouer une musique d'ambiance. La manipulation d'une musique se fait par le type `Mix_Music *` et s'ouvre par l'appel à la fonction `Mix_LoadMUS` depuis le chemin d'une musique. Une musique devra être libérée via `Mix_FreeMusic`.

```
Mix_Music * musique = NULL;
if((musique = Mix_LoadMUS("media/music.mp3")) == NULL) {
    fprintf(stderr, "Erreur d'ouverture de \"media/music.mp3\" :
    %s\n", Mix_GetError());
    exit(EXIT_FAILURE);
}

/* Utilisation de la musique */

Mix_FreeMusic(musique);
musique = NULL;
```

20.2.2 Lecture

La musique peut ensuite être lue par l'appel à `Mix_PlayingMusic`. Celle-ci prend en arguments la musique à jouer et le nombre de répétitions de celle-ci (-1 pour la répéter indéfiniment) :

```
/* Joue musique sans la répéter. */
Mix_PlayMusic(musique, 0);
```

20.2.3 Volume

Le volume de la musique peut être réglé par `Mix_VolumeMusic` prenant un argument de volume sonore. À noter que la constante `MIX_MAX_VOLUME` défini le volume maximum de la musique actuellement jouée.

```
/* Joue musique indéfiniment à 80% du volume maximum. */
Mix_PlayMusic(musique, -1);
Mix_VolumeMusic(MIX_MAX_VOLUME * 0.80);
```

20.2.4 Fin de musique

Il est possible de terminer la musique actuelle en fondu via `Mix_FadeOutMusic` prenant en argument la durée de fondu en millisecondes.

```
/* La musique s'arrête progressivement en 2 secondes. */
Mix_FadeOutMusic(2000);
```

Ceci peut par exemple être utilisé pour jouer un musique de Boss. Pour déclencher un action à la fin de la musique (fin naturelle ou fermée en fondu), il est possible d'enregistrer une fonction auprès de `Mix_HookMusicFinished`.

```
/* Fonction pour jouer la musique du Boss */
void jouerMusiqueBoss() {
    Mix_RewindMusic();
    Mix_PlayMusic(musiqueBoss, -1);
}
```

```
/* Instructions pour enregistrer l'action de jouer la musique du
→ Boss */
Mix_HookMusicFinished(jouerMusiqueBoss);
/* Terminer progressivement la musique actuelle */
if(Mix_PlayingMusic()) {
    Mix_FadeOutMusic(2000);
} else {
    jouerMusiqueBoss();
}
```

20.2.5 Contrôler la musique

Mettre la musique en pause

La musique actuelle peut être mise en pause avec `Mix_PauseMusic()` et reprise via `Mix_ResumeMusic()`. À noter qu'il est possible de savoir si une musique est en pause via `Mix_PausedMusic` :

```
/* Arrête ou relance la musique actuelle : */
if(! Mix_PausedMusic()) {
    Mix_PauseMusic();
} else {
```

```
Mix_ResumeMusic();  
}
```

Arrêter la lecture et rembobiner

La musique actuelle peut être arrêtée par `Mix_HaltMusic` et le lecteur rembobiné par `Mix_RewindMusic` pour jouer une autre musique par exemple.

```
Mix_HaltMusic()  
Mix_RewindMusic();  
Mix_PlayMusic(musiqueBoss, -1);
```

Se rendre à un moment précis de la musique

À noter qu'il est possible de choisir la moment de la musique où se rendre via `Mix_SetMusicPosition` par la position temporelle souhaitée en secondes :

```
/* Se rend à la première minute de la musique : */  
Mix_SetMusicPosition(60.);
```

20.3 Canaux

20.3.1 Mise en place

Pour jouer dans sons, il est nécessaire de définir des canaux dans lesquels les jouer. Ceci se fait par la fonction `Mix_AllocateChannels` et un nombre de canaux. En général 8 canaux est une bonne idée et est la valeur de `MIX_CHANNELS`.

```
Mix_AllocateChannels(MIX_CHANNELS);
```

20.3.2 Jouer des sons

Les sons vont se modéliser par le type `Mix_Chunk *` : ceci permet de charger des sons courts. Il est possible de les charger depuis un fichier via la fonction `Mix_LoadWAV` et un chemin vers un fichier `.via`. Il faudra naturellement libérer le son via `Mix_FreeChunk`.

```

Mix_Chunk * son = NULL;
if((son = Mix_LoadWAV("media/bruit.wav")) == NULL) {
    fprintf(stderr, "Erreur d'ouverture de \"media/bruit.wav\""
    ↵ : "%s\n", Mix_GetError());
    exit(EXIT_FAILURE);
}

/* Instructions d'utilisation */

Mix_FreeChunk(son);
son = NULL;

```

Le son peut ensuite être joué dans un canal via `Mix_PlayChannel` :

- `int channel` : canal à paramétriser (-1 pour un canal libre : à éviter).
- `Mix_Chunk * chunk` : son à jouer.
- `int loops` : répétition du morceau.

Un conseil pourrait être de dédier un canal à un certain type de son : par exemple un pour le bruit des adversaires (bruit spatialisé), un autre pour les bruits du personnage du joueur (bruit non spatialisé).

```
Mix_PlayChannel(1, son, 0);
```

À noter que le volume d'un peu peut se paramétriser par `Mix_VolumeChunk`. De même le volume du canal peut aussi se paramétriser par `Mix_Volume`.

De même des fonctionnalités vues pour la musique sont aussi disponibles pour les canaux :

- `void Mix_Pause(int channel)` : Pour mettre en pause un canal (ou tous via -1).
- `void Mix_Resume(int channel)` : Pour reprendre le son d'un canal (ou tous via -1).
- `int Mix_Paused(int channel)` : Pour déterminer si un canal est en pause (ou -1 pour savoir s'il en existe un).

- `int Mix_Playing(int channel)` : Indique si un son est actuellement joué.
- `Mix_Chunk * Mix_GetChunk(int channel)` : Renvoie le son actuellement joué.
- `int Mix_HaltChannel(int channel)` : Arrête un canal (ou tous via -1).
- `int Mix_FadeOutChannel(int which, int ms)` : Réduit progressivement le son d'un canal.

20.3.3 Paramétrage spatial

Pour offrir plus de réalisme, SDL Mixer propose pouvoir jouer sur la spatialisation (répartition stéréo du son et volume) en fonction d'une donnée d'angle et de distance pour un canal via `Mix_SetPosition` :

- `int channel` : canal à paramétriser.
- `Sint16 angle` : angle en degrés.
- `Uint8 distance` : distance entre 0 (utilisateur) et 255 (inaudible).

21 Parallélisme : SDL Thread

Pour certaines opérations, il peut être intéressant de séparer les processus : un qui gère les événements de l'utilisateur, un autre qui gère l'affichage, un autre une simulation de jeu, un autre le réseau et autres. `SDL_Thread` propose un usage simplifié des threads. Ils seraient à étudier plus en détail dans un cours de programmation concurrente.

21.1 Mise en place

L'utilisation de `SDL Thread` est intégrée à la base de `SDL` et ne demande que l'import de son entête `SDL/SDL_thread.h`.

21.2 Lancer un thread

Un thread est géré par le type `SDL_Thread *`. Lorsqu'il est lancé via la fonction `SDL_CreateThread`, il a besoin d'une fonction de `int (*)(void *)` à exécuter et d'une donnée utilisateur optionnelle. Dans le reste du code, il est ensuite possible d'attendre que la fonction termine de s'exécuter sur le thread et récupérer sa valeur de retour par `SDL_WaitThread` :

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL_thread.h>

int process(void * data) {
    printf("data = %d\n", *((int *)data));
    return 0;
}

int main() {
    int val = 42;
    SDL_Thread * thread = SDL_CreateThread(process, &val);
    printf("Thread lancé.\n");
    int res;
    SDL_WaitThread(thread, &res);
```

```

    printf("Thread terminé avec retour %d\n", res);
    exit(EXIT_SUCCESS);
}

```

À noter que si un thread devait durer trop longtemps ou nécessite d'être terminé, `SDL_KillThread` permet son arrêt.

```

#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>

int process(void * data) {
    printf("data = %d\n", *((int *)data));
    SDL_Delay(10000);
    printf("fini.\n");
    return 0;
}

int main() {
    int val = 42;
    SDL_Thread * thread = SDL_CreateThread(process, &val);
    printf("Thread lancé.\n");
    SDL_Delay(100);
    SDL_KillThread(thread);
    printf("Thread terminé\n");
    exit(EXIT_SUCCESS);
}

```

21.3 Partager des ressources

L'une des problématiques de l'utilisation de threads est le partage de ressources. En effet, plusieurs threads peuvent accéder à une même ressource et leurs traitements étant individuels se chevauchant, ils rendraient incohérente la ressource. Pour palier à cette problématique, il est possible de verrouiller un morceau de code à l'aide d'une `SDL_mutex`. Le déclenchement de l'accès se fait via `SDL_mutexP` :

- Dans le cas où le verrou est actif, le programme attend jusqu'à réussir à saisir le passage.

- Le passage est réalisé que lorsque le verrou était inactif et devient actif une fois réalisé.

La ressource pourrait être libérée via `SDL_mutexV`.

```
/* Création d'un verrou : */
SDL_mutex * verrou = SDL_CreateMutex();
/* Accès protégé à la ressource partagée : */
SDL_mutexP(verrou);
/* Utilisation de la ressource partagée. */
/* Libération du verrou d'accès à la ressource partagée : */
SDL_mutexV(verrou);
```

Ce concept permet au code suivant de fonctionner (mettre en commentaire les appels de `SDL_mutexP` et `SDL_mutexV` pour constater l'incohérente de la donnée utilisée) :

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>

SDL_mutex * verrou = NULL;
int val = 42;

int first(void * data) {
    int * val = (int *)data;
    int i, j;
    for(i = 0; i < 10; ++i) {
        SDL_mutexP(verrou);
        *val = 1;
        for(j = 0; j < 41; ++j) {
            ++(*val);
            SDL_Delay(1);
        }
        printf("first : %d\n", *val);
        SDL_mutexV(verrou);
        SDL_Delay(1);
    }
    return 0;
}

int second(void * data) {
```

```
int * val = (int *)data;
int i, j;
for(i = 0; i < 10; ++i) {
    SDL_mutexP(verrou);
    *val = 1;
    for(j = 0; j < 41; ++j) {
        ++(*val);
        SDL_Delay(1);
    }
    printf("second : %d\n", *val);
    SDL_mutexV(verrou);
    SDL_Delay(1);
}
return 0;
}

int main() {
verrou = SDL_CreateMutex();
SDL_Thread * firstT = SDL_CreateThread(first, &val);
SDL_Thread * secondT = SDL_CreateThread(second, &val);
int i, j;
for(i = 0; i < 10; ++i) {
    SDL_mutexP(verrou);
    val = 1;
    for(j = 0; j < 41; ++j) {
        ++val;
        SDL_Delay(1);
    }
    printf("main : %d\n", val);
    SDL_mutexV(verrou);
    SDL_Delay(1);
}
SDL_WaitThread(firstT, NULL);
SDL_WaitThread(secondT, NULL);
exit(EXIT_SUCCESS);
}
```

21.4 Gestion parallèle d'événements, d'affichage et d'une simulation

Dans un cas plus pratique, ce concept pourrait être utilisé pour séparer les différents processus de notre jeu : gestion des événements, processus d'affichage à l'écran et simulation / calculs en jeu. Ceci est illustré sur une entité subissant de la physique par l'exemple de code ci-dessous :

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>

double px = 300, py = 300;
double vx = 0, vy = 0;
double ax = 0, ay = 0;
int need_refresh = 1;

SDL_mutex * refreshMutex = NULL;
SDL_mutex * positionMutex = NULL;
SDL_mutex * accMutex = NULL;

int affichage(void * data) {
    SDL_Surface * ecran = (SDL_Surface *)data;

    for(;;) {
        SDL_mutexP(refreshMutex);
        if(! need_refresh) {
            SDL_mutexV(refreshMutex);
            continue;
        }
        need_refresh = 0;
        SDL_mutexV(refreshMutex);
        SDL_mutexP(positionMutex);
        SDL_Rect position;
        position.x = px - 10;
        position.y = py - 10;
        SDL_mutexV(positionMutex);
        position.w = 20;
        position.h = 20;
        SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 127, 127,
→ 127));
    }
}
```

```

        SDL_FillRect(ecran, &position, SDL_MapRGB(ecran->format, 25,
        ↳ 25, 25));
        SDL_Flip(ecran);
        SDL_Delay(1000 / 60);
    }
}

int events(void * data) {
    SDL_Event event;
    int active = 1;
    while(active) {
        SDL_WaitEvent(&event);
        switch(event.type) {
            case SDL_QUIT : active = 0; break;
            case SDL_KEYUP : {
                switch(event.key.keysym.sym) {
                    case SDLK_ESCAPE : active = 0; break;
                    case SDLK_LEFT : {
                        SDL_mutexP(accMutex);
                        ax -= 500.;
                        ay -= 1000.;
                        SDL_mutexV(accMutex);
                    } break;
                    case SDLK_RIGHT : {
                        SDL_mutexP(accMutex);
                        ax += 500.;
                        ay -= 1000.;
                        SDL_mutexV(accMutex);
                    } break;
                    default : break;
                }
            } break;
            default : break;
        }
    }
    return 0;
}

int process(void * data) {
    double step = 0.1;
    for(;;) {

```

```
vx += step * ax;
vy += step * ay;
SDL_mutexP(positionMutex);
px += step * vx;
py += step * vy;
if(px > 500) {
    px = 500;
    vx *= -0.5;
}
if(py > 500) {
    py = 500;
    vy *= -0.5;
}
if(px < 100) {
    px = 100;
    vx *= -0.5;
}
if(py < 100) {
    py = 100;
    vy *= -0.5;
}
SDL_mutexV(positionMutex);
SDL_mutexP(accMutex);
ax = 0.;
ay = 9.81;
SDL_mutexV(accMutex);
SDL_mutexP(refreshMutex);
need_refresh = 1;
SDL_mutexV(refreshMutex);
SDL_Delay(1000 / 100);
}
}

int main() {

refreshMutex = SDL_CreateMutex();
positionMutex = SDL_CreateMutex();
accMutex = SDL_CreateMutex();

if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
    fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
}
```

```
    exit(EXIT_FAILURE);
}
SDL_Surface * ecran = NULL;
if((ecran = SDL_SetVideoMode(600, 600, 32, SDL_HWSURFACE |
→ SDL_DOUBLEBUF)) == NULL) {
    fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
    → SDL_GetError());
    SDL_Quit();
    exit(EXIT_FAILURE);
}
SDL_WM_SetCaption("Initiation à la SDL !", NULL);

SDL_Thread * affichageT = SDL_CreateThread(affichage, ecran);
SDL_Thread * eventsT = SDL_CreateThread(events, ecran);
SDL_Thread * processT = SDL_CreateThread(process, ecran);

int res;
SDL_WaitThread(eventsT, &res);
SDL_KillThread(affichageT);
SDL_KillThread(processT);

SDL_FreeSurface(ecran);
SDL_Quit();
exit(EXIT_SUCCESS);
}
```

22 Réseau : SDL Net

Le réseau est une partie qui peut à la fois être intéressante pour permettre de jouer à plusieurs joueurs ou utiliser votre application pour communiquer, mais qui est en général pénible à mettre en place. Vous trouverez donc une alternative simplifiée via `SDL_Net` pour faire communiquer votre programme en réseau.

22.1 Mise en place

La compilation avec `SDL Net` nécessite le linkage de `-lSDL_net` dans la commande de compilation et son fichier d'entête `SDL/SDL_net.h`.

`SDL Net` s'initialise avec `SDLNet_Init` et se termine par `SDLNet_Quit` :

```
if(SDLNet_Init() < 0) {
    fprintf(stderr, "Error in SDLNet_Init: %s\n",
            → SDLNet_GetError());
    exit(EXIT_FAILURE);
}

/* Instructions */

SDLNet_Quit();
```

Ensuite, vous pourrez utiliser la communication via TCP ou via UDP. Grossièrement, la différence est que TCP garanti le bon acheminement des paquets envoyés alors qu'UDP n'offre aucune garantie.

22.2 Avec TCP

22.2.1 Initialisation

L'utilisation de TCP se fait par une première étape de résolution de l'adresse IP vers laquelle se lier ou étape d'indication de que nous jouerons le rôle de serveur. Ceci se fait par l'appel à `SDLNet_ResolveHost` :

- `IPEndPoint * address` : adresse IP à initialiser / affecter.

- `const char * host` : hôte à cible ou `NULL` pour jouer le rôle de serveur.
- `Uint16 port` : port où se connecter (valeur arbitraire).

Ensuite, l'adresse IP obtenue peut être utilisée pour ouvrir un serveur ou ouvrir une connexion vers la cible donnée via `SDLNet_TCP_Open`. La connexion devra plus tard être fermée via `SDLNet_TCP_Close`.

22.2.2 Initialisation serveur

L'initialisation d'un serveur pourrait donner le code suivant :

```
IPAddress ip;
/* Ciblage de la création d'un serveur au port 1337 : */
if(SDLNet_ResolveHost(&ip, NULL, 1337) != 0) {
    fprintf(stderr, "Error in SDLNet_ResolveHost: %s\n",
            SDLNet_GetError());
    exit(EXIT_FAILURE);
}

TCPsocket serveur;
/* Ouverture du serveur : */
if((serveur = SDLNet_TCP_Open(&ip)) == NULL) {
    fprintf(stderr, "Erreur SDLNet_TCP_Open : %s\n",
            SDLNet_GetError());
    exit(EXIT_FAILURE);
}
```

22.2.3 Initialisation client

L'initialisation d'un client pourrait utiliser le code suivant :

```
IPAddress ip;
/* Ciblage d'une machine distante au port 1337 : */
/* Remplacer "localhost" par le nom de la machine */
if(SDLNet_ResolveHost(&ip, "localhost", 1337) != 0) {
    fprintf(stderr, "Error in SDLNet_ResolveHost: %s\n",
            SDLNet_GetError());
    exit(EXIT_FAILURE);
}

/* Affichage de l'adresse IP de la cible : */
```

```

Uint32 intip = SDL_SwapBE32(ip->host);
printf("Target IP is %s at %d.%d.%d.%d:%u\n",
→  SDLNet_ResolveIP(&ip), intip >> 24, (intip >> 16) & 0xff,
→  (intip >> 8) & 0xff, intip & 0xff, ip->port);

TCPsocket serveur;
/* Connexion à la machine distante : */
if((serveur = SDLNet_TCP_Open(&ip)) == NULL) {
    fprintf(stderr, "Erreur SDLNet_TCP_Open : %s\n",
→  SDLNet_GetError());
    exit(EXIT_FAILURE);
}

```

22.2.4 Acceptation d'un client par le serveur

Une fois le serveur mis en place, nous voudrions qu'il puisse se connecter avec un ou plusieurs clients. Pour ceci, il est possible d'accepter une connexion via la fonction `SDLNet_TCP_Accept` prenant en argument le `TCPsocket` ouvert comme serveur et renvoyant un `TCPsocket` correspondant au client associé. Une idée pour accepter plusieurs clients peut être de lancer un `SDL Thread` qui s'occuperait du client accepté :

```

TCPsocket client;
/* En attente de clients : */
for(;;) {
    /* Lorsqu'un client cherche à se connecter, l'accepter : */
    if((client = SDLNet_TCP_Accept(serveur)) == NULL) {
        /* Si aucun client, attendre à nouveau. */
        SDL_Delay(100);
        continue;
    }
    /* Lancer un thread dédié au client : */
    SDL_Thread * thread = SDL_CreateThread(processClient, client);
}

```

À noter que `SDLNet_TCP_Accept` n'est pas bloquant, s'il n'y a pas de client en attente, le retour de la fonction sera `NULL`, c'est à vous de réguler l'attente.

À noter qu'une fois connecté à un client, il est possible d'obtenir son adresse IP via la fonction `IPaddress * SDLNet_TCP_GetPeerAddress(TCPsocket)` sur le `TCPsocket` obtenu.

22.2.5 Échange d'informations

Une fois la relation mise en place, il est possible d'envoyer et recevoir de l'information sur le canal entre les deux machines.

Envoi

L'envoi d'information se fait via `SDLNet_TCP_Send` :

- `TCPsocket sock` : socket depuis lequel envoyer les données.
- `const void * data` : pointeur sur les données à envoyer.
- `int len` : taille en octets des données à envoyer.

Cette fonction renverra `len` dans le cas où les données ont pu être envoyées. Sinon ceci induit une connexion fermée ou une erreur inconnue.

Réception

La réception d'information se fait via `SDLNet_TCP_Recv` :

- `TCPsocket sock` : socket depuis lequel envoyer les données.
- `void * data` : pointeur vers une plage allouée où écrire les données reçues.
- `int maxlen` : taille maximale des données à écrire dans `data`.

Cette fonction renverra le nombre d'octets écrits dans `data`.

22.3 Exemple d'utilisation pour un mini-jeux

Nous pouvons maintenant améliorer le programme multi-threadé proposé précédemment en reliant l'application à un serveur. Nous proposons ici un rapide exemple avec mise en place d'un serveur acceptant les clients en TCP. L'application peut s'illustrer par les codes client et serveur suivants :

```
/* +-----+ */
/* / client.c / */
/* +-----+ */
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_thread.h>
```

```
#include <SDL/SDL_net.h>

#define MAX_PLAYERS 10

double px = 300, py = 300;
double vx = 0, vy = 0;
double ax = 0, ay = 0;
int need_refresh = 1;

int nb_players = 0;
int player_id = -1;
double p[MAX_PLAYERS][2];

SDL_mutex * refreshMutex = NULL;
SDL_mutex * positionMutex = NULL;
SDL_mutex * accMutex = NULL;
SDL_mutex * allMutex = NULL;

int affichage(void * data) {
    SDL_Surface * ecran = (SDL_Surface *)data;

    for(;;) {
        SDL_mutexP(refreshMutex);
        if(! need_refresh) {
            SDL_mutexV(refreshMutex);
            continue;
        }
        need_refresh = 0;
        SDL_mutexV(refreshMutex);
        SDL_mutexP(positionMutex);
        SDL_Rect position;
        position.x = px - 10;
        position.y = py - 10;
        SDL_mutexV(positionMutex);
        position.w = 20;
        position.h = 20;
        SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 127, 127,
            127));

        SDL_Rect positions[MAX_PLAYERS];
        int playersToDraw = 0;
```

```

SDL_mutexP(allMutex);
playersToDraw = nb_players;
int i;
for(i = 0; i < nb_players; ++i) {
    positions[i].x = p[i][0] - 10;
    positions[i].y = p[i][1] - 10;
    positions[i].w = 20;
    positions[i].h = 20;
}
SDL_mutexV(allMutex);
for(i = 0; i < playersToDraw; ++i) {
    SDL_FillRect(ecran, positions + i, SDL_MapRGB(ecran->format,
        → 50, 50, 50));
}

SDL_FillRect(ecran, &position, SDL_MapRGB(ecran->format, 25,
    → 25, 25));

SDL_Flip(ecran);
SDL_Delay(1000 / 60);
}
}

int events(void * data) {
    SDL_Event event;
    int active = 1;
    while(active) {
        SDL_WaitEvent(&event);
        switch(event.type) {
            case SDL_QUIT : active = 0; break;
            case SDL_KEYUP : {
                switch(event.key.keysym.sym) {
                    case SDLK_ESCAPE : active = 0; break;
                    case SDLK_LEFT : {
                        SDL_mutexP(accMutex);
                        ax -= 500.;
                        ay -= 1000.;
                        SDL_mutexV(accMutex);
                    } break;
                    case SDLK_RIGHT : {
                        SDL_mutexP(accMutex);

```

```
    ax += 500.;
    ay -= 1000.;
    SDL_mutexV(accMutex);
} break;
default : break;
}
} break;
default : break;
}
}

return 0;
}

int process(void * data) {
double step = 0.1;
for(;;) {
    vx += step * ax;
    vy += step * ay;
    SDL_mutexP(positionMutex);
    px += step * vx;
    py += step * vy;
    if(px > 500) {
        px = 500;
        vx *= -0.5;
    }
    if(py > 500) {
        py = 500;
        vy *= -0.5;
    }
    if(px < 100) {
        px = 100;
        vx *= -0.5;
    }
    if(py < 100) {
        py = 100;
        vy *= -0.5;
    }
    SDL_mutexV(positionMutex);
    SDL_mutexP(accMutex);
    ax = 0.;
    ay = 9.81;
```

```

        SDL_mutexV(accMutex);
        SDL_mutexP(refreshMutex);
        need_refresh = 1;
        SDL_mutexV(refreshMutex);
        SDL_Delay(1000 / 100);
    }
}

void displayIp(IPAddress * ip) {
    Uint32 intip = SDL_SwapBE32(ip->host);
    printf("%d.%d.%d.%d:%u",
        intip >> 24, (intip >> 16) & 0xff, (intip >> 8) & 0xff, intip &
        ~ 0xff,
        ip->port);
}

int network(void * data) {
    if(SDLNet_Init() < 0) {
        fprintf(stderr, "Error in SDLNet_Init: %s\n",
            ~ SDLNet_GetError());
        exit(EXIT_FAILURE);
    }

    IPAddress ip;
    if(SDLNet_ResolveHost(&ip, "localhost", 1337) != 0) {
        fprintf(stderr, "Error in SDLNet_ResolveHost: %s\n",
            ~ SDLNet_GetError());
        exit(EXIT_FAILURE);
    }
    printf("Target IP is %s at ", SDLNet_ResolveIP(&ip));
    displayIp(&ip);
    printf("\n");

    TCPsocket serveur;

    if((serveur = SDLNet_TCP_Open(&ip)) == NULL) {
        fprintf(stderr, "Erreur SDLNet_TCP_Open : %s\n",
            ~ SDLNet_GetError());
        exit(EXIT_FAILURE);
    }
}

```

```
for(;;) {
    unsigned char message[1024];
    double * positions = (double *)message;
    int len = 2 * sizeof(double);
    SDL_mutexP(positionMutex);
    *positions = px;
    ++positions;
    *positions = py;
    SDL_mutexV(positionMutex);

    int res = SDLNet_TCP_Send(serveur, message, len);

    len = SDLNet_TCP_Recv(serveur, message, 1024);
    if(! len) {
        printf("Absent, terminé.\n");
        break;
    }

    SDL_mutexP(allMutex);
    int * nb = (int *)message;
    nb_players = *nb;
    ++nb;
    player_id = *nb;
    ++nb;
    positions = (double *)nb;
    int i;
    for(i = 0; i < nb_players; ++i) {
        p[i][0] = positions[i * 2];
        p[i][1] = positions[i * 2 + 1];
    }
    SDL_mutexV(allMutex);

    SDL_mutexP(refreshMutex);
    need_refresh = 1;
    SDL_mutexV(refreshMutex);

    SDL_Delay(1000 / 30);
}
SDLNet_TCP_Close(serveur);

SDLNet_Quit();
```

```
}

int main() {

    refreshMutex = SDL_CreateMutex();
    positionMutex = SDL_CreateMutex();
    accMutex = SDL_CreateMutex();
    allMutex = SDL_CreateMutex();

    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(600, 600, 32, SDL_HWSURFACE |
        → SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
        → SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption("Initiation à la SDL !", NULL);

    SDL_Thread * affichageT = SDL_CreateThread(affichage, ecran);
    SDL_Thread * eventsT = SDL_CreateThread(events, ecran);
    SDL_Thread * processT = SDL_CreateThread(process, ecran);
    SDL_Thread * networkT = SDL_CreateThread(network, ecran);

    int res;
    SDL_WaitThread(eventsT, &res);
    SDL_KillThread(affichageT);
    SDL_KillThread(processT);
    SDL_KillThread(networkT);

    SDL_FreeSurface(ecran);
    SDL_Quit();
    exit(EXIT_SUCCESS);
}
```

```
/* +-----+ */
/* / serveur.c / */
/* +-----+ */
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_net.h>
#include <SDL/SDL_thread.h>

#define MAX_PLAYERS 10

SDL_mutex * playersMutex = NULL;

int nb_players = 0;
double p[MAX_PLAYERS][2];
Uint32 ips[MAX_PLAYERS];
Uint16 ports[MAX_PLAYERS];

void displayIp(IPAddress * ip) {
    Uint32 intip = SDL_SwapBE32(ip->host);
    printf("%d.%d.%d.%d:%u",
        intip >> 24, (intip >> 16) & 0xff, (intip >> 8) & 0xff, intip &
        ~ 0xff,
        ip->port);
}

int processClient(void *);

int main() {
    playersMutex = SDL_CreateMutex();
    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    if(SDLNet_Init() < 0) {
        fprintf(stderr, "Error in SDLNet_Init: %s\n",
            ~ SDLNet_GetError());
        exit(EXIT_FAILURE);
    }

    IPAddress ip;
```

```
if(SDLNet_ResolveHost(&ip, NULL, 1337) != 0) {
    fprintf(stderr, "Error in SDLNet_ResolveHost: %s\n",
            → SDLNet_GetError());
    exit(EXIT_FAILURE);
}

TCPsocket serveur;
TCPsocket client;

if((serveur = SDLNet_TCP_Open(&ip)) == NULL) {
    fprintf(stderr, "Erreur SDLNet_TCP_Open : %s\n",
            → SDLNet_GetError());
    exit(EXIT_FAILURE);
}

printf("Serveur lancé, en attente de joueurs.\n");

for(;;) {
    if((client = SDLNet_TCP_Accept(serveur)) == NULL) {
        SDL_Delay(100);
        continue;
    }

    SDL_Thread * thread = SDL_CreateThread(processClient, client);
}

SDLNet_Quit();
SDL_Quit();
exit(EXIT_SUCCESS);
}

int processClient(void * data) {
    TCPsocket client = (TCPsocket)data;
    IPaddress * ip;
    if((ip = SDLNet_TCP_GetPeerAddress(client)) == NULL) {
        fprintf(stderr, "Erreur SDLNet_TCP_GetPeerAddress : %s\n",
                → SDLNet_GetError());
        return 1;
    }

    printf("Accepted connexion with ");
}
```

```
displayIp(ip);
printf("\n");

int player_id = -1;
SDL_mutexP(playersMutex);
if(nb_players >= MAX_PLAYERS) {
    SDL_mutexV(playersMutex);
    printf("Trop de joueurs, connexion refusée.\n");
    SDLNet_TCP_Close(client);
    return 0;
}
player_id = nb_players++;
SDL_mutexV(playersMutex);

printf("Player %d at ", player_id);
displayIp(ip);
printf("\n");

Uint32 currentIp = SDL_SwapBE32(ip->host);
Uint16 currentPort = ip->port;

ips[player_id] = currentIp;
ports[player_id] = currentPort;

for(;;) {
    unsigned char message[1024];
    int len = SDLNet_TCP_Recv(client, message, 1024);
    if(!len) {
        printf("Absent, terminé.\n");

        SDL_mutexP(playersMutex);
        if(nb_players >= 1) {
            ips[player_id] = ips[nb_players - 1];
            ports[player_id] = ports[nb_players - 1];
        }
        --nb_players;
        SDL_mutexV(playersMutex);

        break;
    }
    double * positions = (double *)message;
```

```

double x = *positions;
++positions;
double y = *positions;

SDL_mutexP(playersMutex);
if(ips[player_id] != currentIp || ports[player_id] !=
→ currentPort) {
    for(player_id = 0; player_id < nb_players; ++player_id) {
        if(ips[player_id] == currentIp && ports[player_id] ==
→ currentPort) {
            break;
        }
    }
}
p[player_id][0] = x;
p[player_id][1] = y;
SDL_mutexV(playersMutex);

int * nb = (int *)message;

SDL_mutexP(playersMutex);
*nb = nb_players;
++nb;
*nb = player_id;
++nb;
positions = (double *)nb;
int i;
for(i = 0; i < nb_players; ++i) {
    positions[i * 2] = p[i][0];
    positions[i * 2 + 1] = p[i][1];
}
len = 2 * sizeof(int) + 2 * nb_players * sizeof(double);
SDL_mutexV(playersMutex);

int res = SDLNet_TCP_Send(client, message, len);
}
SDLNet_TCP_Close(client);
return 0;
}

```

Sixième partie

Se préparer à l'examen

Table des matières

A Corrections exercices d'entraînement	595
A.1 Introduction	595
A.1.1 Cours	595
A.1.2 Entraînement	596
A.2 Variables	598
A.2.1 Cours	598
A.2.2 Entraînement	599
A.3 Expressions	611
A.3.1 Cours	611
A.3.2 Entraînement	611
A.4 Conditions	624
A.4.1 Cours	624
A.4.2 Entraînement	626
A.5 Boucles	652
A.5.1 Entraînement	652
A.6 Fonctions	673
A.6.1 Cours	673
A.6.2 Entraînement	673
A.7 Tableaux	718
A.7.1 Échauffement	718
A.7.2 Entraînement	719
A.8 Pointeurs	740
A.8.1 Cours	740
A.8.2 Échauffement	740
A.8.3 Consolidation	744
A.8.4 Entraînement	748
A.9 Consolidation des bases	770

A.9.1	Entraînement	770
A.10	Fichiers	795
	A.10.1 Entraînement	795
A.11	Structures	807
	A.11.1 Entraînement	807
A.12	Programmation modulaire	903
	A.12.1 Entraînement	903
A.13	Types génériques	922
	A.13.1 Entraînement	922
A.14	Opérations bit-à-bit	1020
	A.14.1 Entraînement	1020
B	Examens des années précédentes	1029
B.1	Données statistiques partiels 2021 - 2023	1029
	B.1.1 Semestre 1	1030
	B.1.2 Semestre 2	1031
B.2	Sujets partiels semestre 1	1033
	B.2.1 S1 Sujet zéro	1034
	B.2.2 S1 2021-2022 : Alternance	1041
	B.2.3 S1 2021-2022 : Janvier	1047
	B.2.4 S1 2022-2023 : Alternance	1053
	B.2.5 S1 2022-2023 : Janvier	1061
	B.2.6 S1 2023-2024 : Alternance	1067
	B.2.7 S1 2023-2024 : Janvier initial	1077
	B.2.8 S1 2023-2024 : Janvier alternance	1088
B.3	Sujets partiels semestre 2	1098
	B.3.1 S2 Sujet zéro	1099
	B.3.2 S2 2021-2022 : Alternance	1105
	B.3.3 S2 2021-2022 : Janvier	1111
	B.3.4 S2 2022-2023	1116
	B.3.5 S2 2023-2024 : Alternance	1122
	B.3.6 S2 2023-2024 : Janvier initial	1133

A Corrections exercices d'entraînement

A.1 Introduction

A.1.1 Cours

Correction 1 (Formaliser une logique).

Exemple de méthodes de tri avec des instructions élémentaires en français :

Tri par insertion

1. Placer la liste mélangée sur le bureau sous forme d'un tas.
2. Prévoir un espace où placer la liste triée de sorte que les papiers restent visibles.
3. Piocher le premier papier disponible de la liste mélangée.
4. Pour ce papier pioché, parcourir la liste triée de gauche à droite, lorsque le papier suivant est plus grand (ordre numérique), s'arrêter et insérer le papier en main à gauche de ce papier.
5. Si le parcours est arrivé à terme sans trouver de papier plus grand, ajouter le papier en main à la droite du dernier papier.
6. répéter depuis l'étape 3 tant que la liste mélangée n'est pas vide.

Tri par sélection

1. Placer la liste mélangée sur le bureau sous forme d'un tas.
2. Prévoir un espace où placer la liste triée de sorte que les papiers restent visibles.
3. Piocher le premier papier disponible de la liste mélangée.
4. Parcourir tous les papiers de la liste mélangée et échanger la valeur en main avec celle regardée si celle regardée est plus petite.
5. Placer le papier en main à la fin de la liste triée.
6. répéter depuis l'étape 3 tant que la liste mélangée n'est pas vide.

Tri à bulle

1. Étaler horizontalement la liste mélangée.
2. Parcourir la liste de gauche à droite en regardant les valeurs deux à deux :
3. Pour ces valeurs, lorsque la valeur à gauche est plus grande que la valeur à droite, échanger ces valeurs et répéter si besoin sur les couples suivants pendant le parcours.
4. répéter depuis l'étape 2 tant qu'il existe deux valeurs dans la liste dont la précédente est plus grande que la suivante.

L'idée de l'exercice était de mettre en évidence que bien que le concept puisse sembler simple à imaginer (trier une liste de valeurs) et comprendre, expliquer ce concept à quelqu'un et le formaliser à l'écrit pour que tout personne puisse le réaliser à l'aide d'opérations élémentaires telles qu'elles pourraient être données à une machine est un processus plus périlleux même en langage connu.

Correction 2 (★ Compiler Hello ESGI!).

Se référer aux instructions d'installation d'un compilateur (`gcc` ou `clang`) en fonction du système d'exploitation choisi pour suivre le cours ou à une alternative fonctionnelle. En cas d'échec et d'impossibilité d'installer un compilateur : utiliser **Online GDB**.

Pour compiler via le terminal :

- Se rendre à l'emplacement depuis lequel nous souhaitons compiler (par exemple là où se trouve le fichier source) :

```
cd "cours/01_introduction"  
gcc -o prog hello_esgi.c  
./prog
```

A.1.2 Entrainement**Correction 3 (★ Votre Hello ESGI).**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     printf("ESGI : Bachelor premiere annee\n");
6     printf("Seance 1\n");
7     printf("TRANCHO Kevin\n");
8     exit(EXIT_SUCCESS);
9 }
```

A.2 Variables

A.2.1 Cours

Correction 4 (★ Déclarer et définir des variables).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int ent1;
6     float vir1;
7     int ent2 = 42;
8     char chr1 = '@';
9     vir1 = 13.37;
10    ent1 = ent2;
11    exit(EXIT_SUCCESS);
12 }
```

Correction 5 (★★ Afficher des variables).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     float reel = 13.37;
6     char caractere = 'Z';
7     long entier = 42000000000;
8
9     printf("reel : %g\n", reel);
10    printf("caractere : '%c'\n", caractere);
11    printf("caractere : %d\n", caractere);
12    printf("entier : %ld\n", entier);
13    printf("entier : %lx\n", entier);
14    exit(EXIT_SUCCESS);
15 }
```

Correction 6 (★★ Lire et afficher une variable).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     double reel;
7
8     printf("Entrez un réel : ");
9     scanf("%lf", &reel);
10    printf("Voici votre réel : %g\n", reel);
11
12    exit(EXIT_SUCCESS);
13 }
```

A.2.2 Entrainement

Correction 7 (★ Affichage formaté hexadécimal).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     unsigned long entier;
6     printf("Entrez un entier : ");
7     scanf("%lu", &entier);
8     printf("%lu = 0x%016lX\n", entier, entier);
9     exit(EXIT_SUCCESS);
10 }
```

Correction 8 (★ Affectations).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     char car;
6     /* créer un entier ent */
7     int ent;
```

```
8  /* mettre '4' dans car */
9  car = '4';
10 /* mettre 2 dans ent */
11 ent = 2;
12 /* afficher le caractère car et entier */
13 printf("%c%d\n", car, ent);
14 /* afficher l'entier correspondant à car */
15 printf("%d\n", car);
16 exit(EXIT_SUCCESS);
17 }
```

Correction 9 (★ Traduction en Langage C).

```
1  /* Code Alice : import studio */          */
2  /* Potentielle idée d'importation des fonctionnalités */      */
3  /* standards : */                         */
4  /* - stdio : standard inputs and outputs / entrées / sorties */ */
5  /* - stdlib : standard library */           */
6  #include <stdio.h>                      */
7  #include <stdlib.h>                     */

8
9  /* main est le point d'entrée d'un programme en langage C */      */
10 /* (première fonction lancée). */                                     */
11 /* On l'écrit "int main ()" car c'est en réalité une fonction */ */
12 /* qui renvoie un entier (code d'erreur) et ici ne prend pas */ */
13 /* d'arguments. */          */
14 int main() {
15     /* Code Alice : pi <- 3,14 */                                */
16     float pi;                                         */
17     /* Pour utiliser une variable en langage C, il faut la */ */
18     /* déclarer. */                                     */
19     pi = 3.14;                                         */
20     /* en langage C, la notation numérique est à l'anglaise, on */ */
21     /* utilise un '.' */                               */
22     /* Code Alice : print(pi) */                           */
23     /* Code Alice : Pourquoi ça n'affiche pas pi ??? */ */
24     printf("%f\n", pi);                                     */
25     /* pi ne s'affiche pas car print n'est pas une fonction */ */
26     /* connue en C. */                                 */
27     /* La fonction à utiliser est printf. */             */
```

```

28  /* printf requiert une chaîne avec indication de formats      */
29  /* commençant par % pour associer à chaque % une des valeurs */
30  /* listées après la chaîne (dans le même ordre). */
31  printf("%.1f\n", pi);
32  /* Pour choisir un nombre de décimales / chiffres après la   */
33  /* virgule, il faut ajouter ce nombre précédé d'un '.' dans   */
34  /* le format après le %.                                         */
35  printf("%g\n", pi);
36  /* %g propose automatiquement un affichage convenable pour   */
37  /* un œil humain.                                              */
38  printf("%e\n", pi);
39  /* %e affiche sous notation scientifique.                      */
40  exit(EXIT_SUCCESS);
41  /* exit prend un argument qui correspond à un code d'erreur : */
42  /* EXIT_SUCCESS indique que le programme s'est terminé en    */
43  /* fonctionnant tel qu'il attendait de l'être.                  */
44  /* Note : dans le cas particulier de la fonction main :       */
45  /* /* return 0 aurait pu être utilisé.                         */
46 }

```

Correction 10 (★★ Dépassement capacité d'un int).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5  /* Code Bob : int gros_nombre;                                */
6  unsigned long gros_nombre;
7  /* 1. En général, unsigned long donne un entier sous 8      */
8  /* octets sans valeurs négatives et est donc le plus grand */
9  /* type d'entiers atomiques.                                 */
10 /* Sous un compilateur 32 bits (selon votre installation, */
11 /* principalement sous Windows), il faudra utiliser unsigned */
12 /* long long pour passer sur 8 octets.                      */
13 printf("Entrez un gros nombre : ");
14 /* Code Bob scanf("%d", gros_nombre);                          */
15 scanf("%lu", &gros_nombre);
16 /* Bob doit utiliser le format %lu pour lire un unsigned   */
17 /* long (respectivement %llu pour unsigned long long).     */
18 /* A noter que Bob a oublié de passer gros_nombre par son */

```

```
19  /* adresse Gros_nombre . */  
20  /* Code Bob : printf("%d, un gros nombre ?\n"); */  
21  printf("%lu, un gros nombre ?\n", gros_nombre);  
22  /* Bob a oublié de lister la valeur de gros_nombre après la */  
23  /* chaîne. */  
24  /* Le format d'affichage est de même %lu */  
25  /* (respectivement %llu). */  
26  gros_nombre = 99999999999999999999;  
27  /* Code Bob : printf("%d, un gros nombre !\n"); */  
28  printf("%lu, un gros nombre !\n", gros_nombre);  
29  gros_nombre = 0xffffffffffffffffffff;  
30  /* 2. */  
31  printf("%lu\n", gros_nombre);  
32  /* 3. */  
33  printf("%lx\n", gros_nombre);  
34  int entier = 0xffffffffffffffffffff;  
35  /* 4. gros nombre affiché en hexadécimal comme un int */  
36  /* (4 octets). */  
37  printf("%lx\n", entier);  
38  /* int est limité à la fois par le bit de signe et par son */  
39  /* occupation mémoire. */  
40  printf("%016lx\n", entier);  
41  exit(EXIT_SUCCESS);  
42 }
```

Correction 11 (★ Imprécision flottante).

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main() {  
5  /* Pour corriger le code de Charlie, */  
6  /* il suffit de choisir le long double si disponible : */  
7  /* dans le cas de Windows, on peut gagner en précision avec */  
8  /* le double. */  
9  /* Code Charlie : float racine = 1.414213562373095048; */  
10 long double racine = 1.4142135623730950481;  
11 /* Code Charlie : printf("racine de 2 vaut %g\n", racine); */  
12 printf("racine de 2 vaut %Lg\n", racine);  
13 /* Code Charlie : printf("variable : %.18f\n", racine); */
```

```

14 printf("variable : %.18Lf\n", racine);
15 printf("texte     : 1.414213562373095048\n");
16 /* Code Charlie : C'est différent, étrange ... */ 
17 /* Code Charlie : float clavier; */ 
18 long double clavier;
19 printf("Recopiez :\n>1.414213562373095048\n");
20 /* Code Charlie : scanf("%f", &clavier); */ 
21 scanf("%Lf", &clavier);
22 /* Code Charlie : printf("copie : %.18f\n", clavier); */ 
23 printf("copie : %.18Lf\n", clavier);
24 printf("texte : 1.414213562373095048\n");
25 /* Code Charlie : C'est la machine qui bug ! */ 
26 exit(EXIT_SUCCESS);
27 }
```

Correction 12 (★★★ Partie entière d'un flottant).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     long double value;
6     printf("Entrez un réel : ");
7     scanf("%Lf", &value);
8     printf("La partie entière de %Lg est environ %.0Lf\n", value,
9            → value);
10    exit(EXIT_SUCCESS);
11 }
```

Correction 13 (★★★★ Hexadécimal ?).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Sauvegarder fb40 8be9 */
6     unsigned int code = 0xfb408be9;
7     printf("Code : %u\n", code);
```

```
8   printf("%X%d\n", 42, 42);
9   unsigned long first = 212063991488173;
10  unsigned long second = 223196547513038;
11  printf("\if %lx then %lx\n", first, second);
12  exit(EXIT_SUCCESS);
13 }
```

Correction 14 (★★★★★ Un message secret).

```
1 /* +-----+ */
2 /* / FICHIER : code.c / */
3 /* +-----+ */

4
5 #include <stdio.h>
6 #include <stdlib.h>

7
8 int main() {
9     unsigned char first;
10    unsigned char second;
11    unsigned char third;
12    unsigned char fourth;
13    unsigned int entier;
14    float flottant;
15    printf("Entrez quatre caractères : ");
16    scanf("%c%c%c%c", &first, &second, &third, &fourth);
17    entier = first + second * 0x100 + third * 0x10000 + fourth *
18        → 0x1000000;
19    printf("Copiez l'entier %x : ", entier);
20    scanf("%x", &flottant);
21    printf("Codage : %.8e\n", flottant);
22    exit(EXIT_SUCCESS);
23 }
```

```
1 /* +-----+ */
2 /* / FICHIER : decode.c / */
3 /* +-----+ */

4
5 #include <stdio.h>
6 #include <stdlib.h>
```

```

7 int main() {
8     unsigned char first;
9     unsigned char second;
10    unsigned char third;
11    unsigned char fourth;
12    unsigned int entier;
13    float flottant;
14
15    printf("Entrez un réel : ");
16    scanf("%f", &entier);
17    first = entier;
18    second = entier / 0x100;
19    third = entier / 0x10000;
20    fourth = entier / 0x1000000;
21    printf("Texte : \"%c%c%c%c\"\n", first, second, third, fourth);
22    exit(EXIT_SUCCESS);
23 }
```

Correction 15 (∞ Mini-projet : dessins de formes).

```

1  /* +-----+ */
2  /* | FICHIER : main.c | */
3  /* +-----+ */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <math.h>
8  #include <time.h>
9  #include <SDL/SDL.h>
10 #include <SDL/SDL_gfxPrimitives.h>
11
12 /* Paramètres de la fenêtre : */
13 const int largeur = 800;
14 const int hauteur = 800;
15 const char * titre = "ESGI shapes draw";
16
17 int call_command(char shape_id);
18 /* Valeurs attendues entre -1 et 1 pour les coordonnées et 0 et 1
   → pour les couleurs : */
19 void draw_circle(double center_x, double center_y, double rayon,
   → double r, double g, double b);
```

```
20 void draw_line(double start_x, double start_y, double end_x,
→   double end_y, double r, double g, double b);
21
22 void read_circle() {
23     double cx, cy, rayon;
24     double r, g, b;
25     scanf("%lf %lf %lf %lf %lf", &cx, &cy, &rayon, &r, &g, &b);
26     draw_circle(cx, cy, rayon, r, g, b);
27 }
28
29 void read_line() {
30     double sx, sy, ex, ey;
31     double r, g, b;
32     scanf("%lf %lf %lf %lf %lf %lf", &sx, &sy, &ex, &ey, &r, &g,
→   &b);
33     draw_line(sx, sy, ex, ey, r, g, b);
34 }
35
36 int read_command() {
37     char command;
38     printf("Entrez une commande :\n - c : cercle (centre(x, y) rayon
→   couleur(r, g, b))\n - l : ligne (debut(x, y) fin(x, y)
→   couleur(r, g, b))\n - q : quitter\n>>> ");
39     scanf(" %c", &command);
40     return call_command(command);
41 }
42
43 int call_command(char shape_id) {
44     switch(shape_id) {
45         case 'c' : read_circle(); break;
46         case 'l' : read_line(); break;
47         case 'q' : return 0;
48         default : break;
49     }
50     return 1;
51 }
52
53 SDL_Surface * ecran = NULL;
54
55 void draw_circle(double center_x, double center_y, double rayon,
→   double r, double g, double b) {
```

```

56     filledCircleRGBA(ecran, center_x * largeur / 2. + largeur / 2.,
57         -center_y * hauteur / 2. + hauteur / 2.,
58         rayon * fmin(largeur, hauteur) / 2., r * 255, g * 255, b *
59             255, 255);
60 }
61
62 void draw_line(double start_x, double start_y, double end_x,
63     double end_y, double r, double g, double b) {
64     lineRGBAlpha(ecran, start_x * largeur / 2. + largeur / 2., -start_y
65         * hauteur / 2. + hauteur / 2.,
66         end_x * largeur / 2. + largeur / 2., -end_y * hauteur / 2. +
67             hauteur / 2., r * 255, g * 255, b * 255, 255);
68 }
69
70 int main(int argc, char * argv[]) {
71     srand(time(NULL));
72     /* Création d'une fenêtre SDL : */
73     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
74         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
75         exit(EXIT_FAILURE);
76     }
77     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
78         | SDL_DOUBLEBUF)) == NULL) {
79         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
80             SDL_GetError());
81         SDL_Quit();
82         exit(EXIT_FAILURE);
83     }
84     SDL_WM_SetCaption(titre, NULL);
85     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
86         51));
87     SDL_Flip(ecran);
88
89     while(read_command()) {
90
91         SDL_Flip(ecran);
92         SDL_Delay(1000 / 60);
93     }
94
95     SDL_FreeSurface(ecran);
96     SDL_Quit();
97 }
```

```
89     exit(EXIT_SUCCESS);  
90 }
```

```
1  /* +-----+ */  
2  /* / FICHIER : main	SDL2.c / */  
3  /* +-----+ */  
4  
5 #include <stdio.h>  
6 #include <stdlib.h>  
7 #include <math.h>  
8 #include <time.h>  
9 #include <SDL2/SDL.h>  
10 #include <SDL2/SDL2_gfxPrimitives.h>  
11  
12 /* Paramètres de la fenêtre : */  
13 const int largeur = 800;  
14 const int hauteur = 800;  
15 const char * titre = "ESGI shapes draw";  
16  
17 int call_command(char shape_id);  
18 /* Valeurs attendues entre -1 et 1 pour les coordonnées et 0 et 1  
   → pour les couleurs : */  
19 void draw_circle(double center_x, double center_y, double rayon,  
   → double r, double g, double b);  
20 void draw_line(double start_x, double start_y, double end_x,  
   → double end_y, double r, double g, double b);  
21  
22 void read_circle() {  
23     double cx, cy, rayon;  
24     double r, g, b;  
25     scanf("%lf %lf %lf %lf %lf", &cx, &cy, &rayon, &r, &g, &b);  
26     draw_circle(cx, cy, rayon, r, g, b);  
27 }  
28  
29 void read_line() {  
30     double sx, sy, ex, ey;  
31     double r, g, b;  
32     scanf("%lf %lf %lf %lf %lf %lf", &sx, &sy, &ex, &ey, &r, &g,  
   → &b);  
33     draw_line(sx, sy, ex, ey, r, g, b);
```

```

34 }
35
36 int read_command() {
37     char command;
38     printf("Entrez une commande :\n - c : cercle (centre(x, y) rayon
39         → couleur(r, g, b))\n - l : ligne (debut(x, y) fin(x, y)
40         → couleur(r, g, b))\n - q : quitter\n>> ");
41     scanf(" %c", &command);
42     return call_command(command);
43 }
44
45 int call_command(char shape_id) {
46     switch(shape_id) {
47         case 'c' : read_circle(); break;
48         case 'l' : read_line(); break;
49         case 'q' : return 0;
50         default : break;
51     }
52     return 1;
53 }
54
55 SDL_Renderer * ecran = NULL;
56
57 void draw_circle(double center_x, double center_y, double rayon,
58     → double r, double g, double b) {
59     filledCircleRGBA(ecran, center_x * largeur / 2. + largeur / 2.,
60         → -center_y * hauteur / 2. + hauteur / 2.,
61         rayon * fmin(largeur, hauteur) / 2., r * 255, g * 255, b *
62         → 255, 255);
63 }
64
65 void draw_line(double start_x, double start_y, double end_x,
66     → double end_y, double r, double g, double b) {
67     lineRGBA(ecran, start_x * largeur / 2. + largeur / 2., -start_y
68         * hauteur / 2. + hauteur / 2.,
69         end_x * largeur / 2. + largeur / 2., -end_y * hauteur / 2. +
70         hauteur / 2., r * 255, g * 255, b * 255, 255);
71 }
72
73 int main(int argc, char * argv[]) {
74     SDL_Window * window = NULL;

```

```
67     srand(time(NULL));
68     /* Création d'une fenêtre SDL : */
69     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
70         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
71         exit(EXIT_FAILURE);
72     }
73     if(SDL_CreateWindowAndRenderer(largeur, hauteur,
74         → SDL_WINDOW_SHOWN, &window, &ecran) != 0) {
75         fprintf(stderr, "Error in SDL_CreateWindowAndRenderer : %s\n",
76         → SDL_GetError());
77         SDL_Quit();
78         exit(EXIT_FAILURE);
79     }
80     SDL_SetWindowTitle(window, titre);
81     SDL_SetRenderDrawColor(ecran, 51, 51, 51, 255);
82     SDL_RenderClear(ecran);
83     SDL_RenderPresent(ecran);
84
85     while(read_command()) {
86
87         SDL_RenderPresent(ecran);
88         SDL_Delay(1000 / 60);
89     }
90
91     SDL_DestroyRenderer(ecran);
92     SDL_DestroyWindow(window);
93     SDL_Quit();
94     exit(EXIT_SUCCESS);
95 }
```

A.3 Expressions

A.3.1 Cours

Correction 16 (★ Calculer pour l'utilisateur).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     float first, second;
6     printf("Entrez deux réels : ");
7     scanf("%f %f", &first, &second);
8
9     printf("%g + %g = %g\n", first, second, first + second);
10    printf("%g - %g = %g\n", first, second, first - second);
11    printf("%g * %g = %g\n", first, second, first * second);
12
13    exit(EXIT_SUCCESS);
14 }
```

A.3.2 Entraînement

Correction 17 (★★ Opérations).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int a, b;
6     /* demander des valeurs pour a et b */  

7     /* printf affiche à l'utilisateur un message significatif */  

8     /* pour lui indiquer qu'il devra entrer deux valeurs */  

9     /* entières. */  

10    printf("Entrez deux entiers : ");  

11    /* La lecture des deux entiers se fait au clavier par scanf */  

12    /* scanf affecte les variables a et b à l'aide de leur */  

13    /* adresse (donnée par la précédence par &) */  

14    scanf("%d %d", &a, &b);
```

```
15  /* afficher l'addition de a et b */  
16  /* l'addition des valeurs connues pour a et b se fait par */  
17  /* l'opération + ce qui est effectué ici est le remplacement */  
18  /* de a et b par leur valeur puis l'opération arithmétique */  
19  /* d'addition des deux valeurs qui donne un résultat de même */  
20  /* type (int). */  
21  printf("%d + %d = %d\n", a, b, a + b);  
22  /* échanger les valeurs de a et de b */  
23  /* une affectation écrase la valeur, nous proposons ici */  
24  /* l'utilisation d'une variable temporaire : */  
25  int tmp; /* / tmp / a / b / */  
26  tmp = a; /* / a / a / b / */  
27  a = b; /* / a / b / b / */  
28  b = tmp; /* / a / b / a / */  
29  /* afficher la soustraction de a et b */  
30  printf("%d - %d = %d\n", a, b, a - b);  
31  long c;  
32  /* affecter à c le résultat de la multiplication de a et b */  
33  /* notons que nous souhaitons ranger le résultat de la */  
34  /* multiplication de a (int : 4 octets) et b (int : 4 */  
35  /* octets) dans c (long : 8 octets). */  
36  /* Or, a * b renvoie un résultat (int : 4 octets) dont nous */  
37  /* perdrions une partie. Une solution peut être de */  
38  /* transformer une opérande en (long : 8 octets). */  
39  /* L'évaluation de a * b se fait alors sur 8 octets avant */  
40  /* d'être affectée. */  
41  c = (long)a * b;  
42  printf("%d * %d = %ld\n", a, b, c);  
43  float d;  
44  /* affecter à d le résultat de la division fractionnaire de */  
45  /* a et b */  
46  /* a et b sont des entier, la division sera donc la division */  
47  /* entière. Pour obtenir une division fractionnaire, une */  
48  /* proposition est la transformation d'une opérande en */  
49  /* flottant. */  
50  d = (float)a / b;  
51  printf("%d / %d = %g\n", a, b, d);  
52  exit(EXIT_SUCCESS);  
53 }
```

Correction 18 (★ Division par zéro).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int x = 0;
6     /* Alice avait un problème de parenthèses : */
7     /* la multiplication est prioritaire, */
8     /* ceci ne calculait pas  $(x - 1)^2 / (x + 1)$  */
9     int y = (x - 1) * (x - 1) / (x + 1);
10    printf("f(%d) = %d\n", x, y);
11    x = 1;
12    y = (x - 1) * (x - 1) / (x + 1);
13    printf("f(%d) = %d\n", x, y);
14    x = 3;
15    y = (x - 1) * (x - 1) / (x + 1);
16    printf("f(%d) = %d\n", x, y);
17    exit(EXIT_SUCCESS);
18 }
```

Correction 19 (★★ Affection d'une addition ?).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     long big = 0;
6     int ajout;
7     printf("big vaut %ld, faites le grossir : ", big);
8     scanf("%d", &ajout);
9     /* Le +++ est une mauvaise pratique même si le langage */ 
10    /* l'accepte. Dans le cas présent, ceci revient aux */ 
11    /* écritures suivantes : */ 
12    /* big = ajout+++big; */ 
13    /* big = ajout++ + big; */ 
14    /* big = ajout + big; */ 
15    /* big = big + ajout; */ 
16    /* big += ajout; */ 
17    big += ajout;
```

```
18     printf("big vaut %ld !\n", big);
19     exit(EXIT_SUCCESS);
20 }
```

Correction 20 (★★ Imprécision et opérations).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     double resultat = 1.;
6     float ajout = 1e-9;
7     printf("resultat = %.15f\n", resultat);
8     printf("on ajoute %.15f\n", ajout);
9     resultat += ajout;
10    printf("resultat = %.15f\n", resultat);
11    /* Pourquoi resultat ne change pas ? */
12    /* L'ajout d'une valeur trop petite peut ne pas être pris en */
13    /* compte à cause de l'approximation des nombres flottants. */
14    /* Une solution est de prendre un type plus précis : double. */
15    exit(EXIT_SUCCESS);
16 }
```

Correction 21 (★★★ Message codé).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     unsigned int p = 4285404239;
6     unsigned int k = 2015201261;
7     unsigned int code;
8     unsigned int message;
9     printf("Entrez un code : ");
10    scanf("%x", &code);
11    message = ((unsigned long)k * code) % p;
12    printf("Message : %x\n", message);
13    exit(EXIT_SUCCESS);
14 }
```

Correction 22 (★★ Afficher sinus et cosinus d'un angle).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int main() {
6     double deg_angle;
7     printf("Entrez un angle : ");
8     scanf("%lf", &deg_angle);
9     /* Conversion de l'angle en radians : */
10    double rad_angle = deg_angle * M_PI / 180.;
11    /* Appel aux fonctions mathématiques cos et sin : */
12    printf("cos(%g°) = %g\n", deg_angle, cos(rad_angle));
13    printf("sin(%g°) = %g\n", deg_angle, sin(rad_angle));
14    exit(EXIT_SUCCESS);
15 }
```

Correction 23 (★★★ Conversion d'une adresse IP).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     unsigned int entier;
6     printf("Entrez un entier : ");
7     scanf("%u", &entier);
8     /* Extraction des octets à l'aide d'un décalage par
    → l'hexadécimal : */
9     /* - diviser par 0x10 permet de tronquer un chiffre en
    → hexadécimal (soit 4 bits). */
10    /* - le reste d'une division par 0x10 permet de récupérer le
    → dernier chiffre en hexadécimal (4 bits). */
11    /* Nous appliquons ce procédé pour récupérer les valeurs des
    → octets (8 bits) dans entier : */
12    printf("%u = %u.%u.%u.%u\n", entier, entier / 0x1000000, (entier
    → / 0x10000) % 0x100, (entier / 0x100) % 0x100, entier %
    → 0x100);
```

```
13
14     unsigned char a, b, c, d;
15     printf("Saisir une adresse IPv4 : ");
16     scanf("%hhu.%hhu.%hhu.%hhu", &a, &b, &c, &d);
17     /* Construction de la valeur sur un entier par décalage par une
18      → puissance de 16^2 = 256 : */
19     entier = a * 0x1000000 + b * 0x10000 + c * 0x100 + d * 0x1;
20     printf("%u.%u.%u.%u = %u\n", a, b, c, d, entier);
21
22     exit(EXIT_SUCCESS);
}
```

Correction 24 (★★★ Suite de Fibonacci par la formule de Binet).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5
6 /* Note : l'utilisation de math.h demande l'ajout de -lm */
7 /* dans la commande de compilation. */
8
9 /* Le calcul de F_90 donne la valeur 2880067194370816120 */
10 /* (Ceci est la valeur exacte et attendue) */
11 /* Le calcul de F_20000 donne une valeur à 4180 chiffres */
12 /* Cette approximation est à un facteur environ 3.25e-17 */
13 /* De la valeur exacte calculée : soit une approximation */
14 /* pertinente obtenue en environ 3.3e-05 secondes. */
15
16 int main() {
17     int n;
18     printf("Entrez n : ");
19     scanf("%d", &n);
20     /* Nous utilisons les mesures de temps clock_t pour */
21     /* prouver l'efficacité du calcul. */
22     clock_t start;
23     clock_t end;
24     start = clock();
25     /* Pour réussir à effectuer les calculs sur un long */
26     /* double : nous utilisons les fonctions sqrtl et */
```

```

27  /* powl de la bibliothèque math. */  

28  long double first = (1. + sqrtl(5.)) / 2.;  

29  long double second = (1. - sqrtl(5.)) / 2.;  

30  long double res = (powl(first, n) - powl(second, n)) /  

31    → sqrtl(5.);  

32  end = clock();  

33  /* Le réglage de 0 chiffres après la virgule permet */  

34  /* d'obtenir une approximation par un entier et le */  

35  /* choix du %f de ne pas avoir d'approximation */  

36  /* scientifique. */  

37  printf("%.OLf\n", res);  

38  printf("In %g\n", (double)(end - start) / CLOCKS_PER_SEC);  

39  exit(EXIT_SUCCESS);  

}

```

Correction 25 (∞ Shooter).

```

1 #include <stdio.h>  

2 #include <stdlib.h>  

3 #include <math.h>  

4 #include <time.h>  

5 #include <SDL/SDL.h>  

6 #include <SDL/SDL_gfxPrimitives.h>  

7  

8 /* Utilisation de SDL : */  

9 /* Compilation Linux avec les flags -lm -lSDL -lSDL_gfx */  

10  

11 /* Paramètres de la fenêtre : */  

12 const int largeur = 800;  

13 const int hauteur = 800;  

14 const char * titre = "ESGI single shooter";  

15  

16 /* Coordonnées du joueur : */  

17 int px, py;  

18 /* Ligne de tir du joueur : */  

19 int slpx, slpy;  

20 /* Vitesse du joueur : */  

21 int pv;  

22  

23 /* Coordonnées de l'adversaire : */

```

```
24 int ax, ay;
25 /* Vitesse de l'adversaire : */
26 int av;
27
28 void joueur_se_dirige_vers(int tx, int ty) {
29     /* On calcule le vecteur de déplacement vers la cible. */
30     int vx = tx - px;
31     int vy = ty - py;
32     /* On calcule la norme de ce vecteur et ignore le      */
33     /* déplacement si trop petit : évite division par 0. */
34     int v = sqrt(vx * vx + vy * vy);
35     if(v < 1) {
36         return ;
37     } else if(v < pv) {
38         /* Cas où le déplacement est plus petit que la      */
39         /* vitesse : permet plus de précision au joueur. */
40         px += vx;
41         py += vy;
42         return;
43     }
44     /* On déplace le joueur d'un pas de sa vitesse.      */
45     px += (pv * vx) / v;
46     py += (pv * vy) / v;
47 }
48
49 void adversaire_se_dirige_vers(int tx, int ty) {
50     /* Déplacement de l'adversaire analogue à celui de      */
51     /* joueur (Voir joueur_se_dirige_vers). */
52     int vx = tx - ax;
53     int vy = ty - ay;
54     int v = sqrt(vx * vx + vy * vy);
55     if(v < 1) {
56         return ;
57     } else if(v < av) {
58         ax += vx;
59         ay += vy;
60         return;
61     }
62     ax += (av * vx) / v;
63     ay += (av * vy) / v;
64 }
```

```

65
66 void joueur REGARDER(int tx, int ty) {
67     /* On calcule le vecteur de direction vers la cible. */
68     int vx = tx - px;
69     int vy = ty - py;
70     int v = sqrt(vx * vx + vy * vy);
71     /* Si la souris est sur le joueur, on place la fin de */
72     /* la ligne sur la position du joueur : non visible. */
73     if(v < 1) {
74         slpx = px;
75         slpy = py;
76         return ;
77     }
78     /* On envoie la position de fin de la ligne hors de la */
79     /* fenêtre (changement d'échelle du vecteur directeur).*/
80     int dv = largeur + hauteur;
81     slpx = px + (dv * vx) / v;
82     slpy = py + (dv * vy) / v;
83 }
84
85 int distance_tir(int x, int y) {
86     /* On prend la direction de la ligne de tir. */
87     int vx = slpx - px;
88     int vy = slpy - py;
89     int v = sqrt(vx * vx + vy * vy);
90     /* Si l'adversaire n'est pas devant le joueur ou collé */
91     /* nous renvoyons une distance invalide : trop loin. */
92     if((x - px) * vx + (y - py) * vy < 0
93     || v < 1) {
94         return 0x7fffffff;
95     }
96     /* Le produit scalaire à un vecteur normalisé donne la */
97     /* distance signée à la droite auquel ce vecteur est */
98     /* perpendiculaire : nous le faisons tourner d'un */
99     /* angle droit pour réaliser ce produit scalaire : */
100    /* (a, b) -> (b, -a). */
101    return abs(vy * (x - px) - vx * (y - py)) / v;
102 }
103
104 void place_adversaire() {
105     /* On prend un angle aléatoire pour placer

```

```

106  /* l'adversaire hors de la fenêtre. */  

107  int angle = rand() % 360;  

108  /* Les distances verticales et horizontales sont de */  

109  /* l'ordre de (largeur + hauteur) / 4. D'après le */  

110  /* théorème de Pythagore, la distance du centre à un */  

111  /* serait de l'ordre de */  

112  /* sqrt(2) * distance(centre, côté) */  

113  /* = sqrt(2) * (largeur + hauteur) / 4 */  

114  /* = 0.35 * (largeur + hauteur). */  

115  int dv = 0.35 * (largeur + hauteur);  

116  /* Note : nous avons la position en coordonnées */  

117  /* polaires (relativement au centre de la fenêtre) et */  

118  /* cos et sin prennent une valeur en radian (d'où la */  

119  /* conversion en radian donnée par (produit en croix) */  

120  /* (angle * 3.1415) / 180 . */  

121  ax = dv * cos((angle * 3.1415) / 180) + largeur / 2;  

122  ay = dv * sin((angle * 3.1415) / 180) + hauteur / 2;  

123 }  

124  

125 int distance_joueur(int x, int y) {  

126  /* Mesure de la distance entre la position du joueur */  

127  /* et la position donnée (norme d'un vecteur / */  

128  /* théorème de Pythagore). */  

129  int vx = x - px;  

130  int vy = y - py;  

131  return sqrt(vx * vx + vy * vy);  

132 }  

133  

134 void affichage(SDL_Surface * ecran) {  

135  /* Remplissage de l'écran par un gris foncé uniforme : */  

136  SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,  

137  → 51));  

138  /* Affichage du joueur : */  

139  filledCircleRGBA(ecran, px, py, 25, 51, 204, 51, 255);  

140  /* Affichage de l'adversaire : */  

141  filledCircleRGBA(ecran, ax, ay, 25, 204, 51, 51, 255);  

142  lineRGBA(ecran, px, py, slpx, slpy, 102, 255, 102, 255);  

143 }  

144  

145 int main() {

```

```
146     srand(time(NULL));
147     /* Création d'une fenêtre SDL : */
148     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
149         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
150         exit(EXIT_FAILURE);
151     }
152     SDL_Surface * ecran = NULL;
153     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
154     → | SDL_DOUBLEBUF)) == NULL) {
155         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
156         → SDL_GetError());
157         SDL_Quit();
158         exit(EXIT_FAILURE);
159     }
160     SDL_WM_SetCaption(titre, NULL);
161
162     /* Placement du joueur au centre de la fenêtre : */
163     px = ecran->w / 2;
164     py = ecran->h / 2;
165     pv = 10;
166
167     ax = ecran->w / 2;
168     ay = 0;
169     av = 5;
170
171     int active = 1;
172     int grab = 0;
173     SDL_Event event;
174     int last_mouse_x = px;
175     int last_mouse_y = py;
176
177     int score = 0;
178     char display[100];
179
180     while(active) {
181
182         affichage(ecran);
183         sprintf(display, "Score : %d", score);
184         stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
185         SDL_Flip(ecran);
186
187     }
188 }
```

```
185     while(SDL_PollEvent(&event)) {
186
187         switch(event.type) {
188             /* Utilisateur clique sur la croix de la fenêtre : */
189             case SDL_QUIT : {
190                 active = 0;
191             } break;
192
193             /* Utilisateur enfonce une touche du clavier : */
194             case SDL_KEYDOWN : {
195                 switch(event.key.keysym.sym) {
196                     /* Touche Echap : */
197                     case SDLK_ESCAPE : {
198                         active = 0;
199                     } break;
200                 }
201             } break;
202
203             /* Utilisateur commence le clic : */
204             case SDL_MOUSEBUTTONDOWN : {
205                 switch(event.button.button) {
206                     case SDL_BUTTON_LEFT : {
207                         grab = 1;
208                         last_mouse_x = event.button.x;
209                         last_mouse_y = event.button.y;
210                     } break;
211
212                     case SDL_BUTTON_RIGHT : {
213                         if(abs(distance_tir(ax, ay)) < 25) {
214                             ++score;
215                             ++pv;
216                             ++av;
217                             place_adversaire();
218                         }
219                     } break;
220                 }
221             } break;
222
223             /* Utilisateur relâche le clic : */
224             case SDL_MOUSEBUTTONUP : {
225                 switch(event.button.button) {
```

```
226         case SDL_BUTTON_LEFT : {
227             grab = 0;
228             } break;
229         }
230     } break;

231
232     /* Utilisateur bouge la souris : */
233     case SDL_MOUSEMOTION : {
234         last_mouse_x = event.motion.x;
235         last_mouse_y = event.motion.y;
236         } break;
237     }
238 }
239 if(grab) {
240     joueur_se_dirige_vers(last_mouse_x, last_mouse_y);
241 }
joueur_regarde(last_mouse_x, last_mouse_y);
adversaire_se_dirige_vers(px, py);
244 if(distance_joueur(ax, ay) < 25) {
245     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 204, 51,
246     ↳ 51));
247     filledCircleRGBA(ecran, px, py, 25, 0, 0, 0, 255);
248     stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
249     SDL_Flip(ecran);
250     SDL_Delay(1000);
251     active = 0;
252 }
253     SDL_Delay(1000 / 60);
254 }

255     SDL_FreeSurface(ecran);
256     SDL_Quit();
257     exit(EXIT_SUCCESS);
258 }
```

A.4 Conditions

A.4.1 Cours

Correction 26 (★ Déterminer une catégorie d'âge).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int age;
6     printf("Quel est votre âge ? ");
7     scanf("%d", &age);
8     printf("Selon le cycle de vie, votre catégorie d'âge est ");
9     if(age <= 14) {
10         printf("Enfant.\n");
11     } else if(age <= 24) {
12         printf("Adolescent.\n");
13     } else if(age <= 64) {
14         printf("Adulte.\n");
15     } else {
16         printf("Aîné.\n");
17     }
18     exit(EXIT_SUCCESS);
19 }
```

Correction 27 (★ Intersection d'événements).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int age;
6     printf("Quel est votre âge ? ");
7     scanf("%d", &age);
8
9     if(age >= 18 && age <= 25) {
10         printf("Vous êtes un jeune de 18 - 25 ans.\n");
11     } else {
```

```

12     printf("Vous n'êtes pas un jeune de 18 - 25 ans.\n");
13 }
14 exit(EXIT_SUCCESS);
15 }
```

Correction 28 (★ Calculer l'amende d'un excès de vitesse).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int vitesse, limitation;
6     int diff;
7     int amende = 0, points = 0;
8     printf("Entrez vitesse et limitation : ");
9     scanf("%d %d", &vitesse, &limitation);
10    diff = vitesse - limitation;
11    if(diff >= 50) {
12        amende = 1500; points = 6;
13    } else if(diff >= 40) {
14        amende = 135; points = 4;
15    } else if(diff >= 30) {
16        amende = 135; points = 3;
17    } else if(diff >= 20) {
18        amende = 135; points = 2;
19    } else if(diff > 0) {
20        if(limitation > 50) {
21            amende = 68; points = 1;
22        } else {
23            amende = 135; points = 1;
24        }
25    }
26    if(diff > 0) {
27        printf("Amende de %d et retrait de %d point%s\n", amende,
28               → points, (points > 1) ? "s" : "");
29    } else {
30        printf("Pas de problème, circulez.\n");
31    }
32    exit(EXIT_SUCCESS);
}
```

A.4.2 Entrainement

Correction 29 (★ Acheter un article).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Nous déclarons les entrées de l'utilisateur : */
6     float argent;
7     float prix;
8     float remise;
9
10    /* Nous lisons les entrées de l'utilisateur : */
11    printf("Votre argent : ");
12    scanf("%f", &argent);
13    printf("Le prix de l'article (hors soldes) : ");
14    scanf("%f", &prix);
15    printf("Remise en %% : ");
16    scanf("%f", &remise);
17
18    /* Nous pouvons rendre conforme les entrées à notre calcul      */
19    /* ou vérifier leur validité :                                     */
20    if(prix < 0.) {
21        printf("Un prix négatif, étrange ... \n");
22        exit(EXIT_FAILURE);
23    }
24    if(remise < 0.) {
25        remise *= -1.;
26    }
27    if(remise > 100.) {
28        printf("La remise ne peut pas excéder -100 %% ... \n");
29        exit(EXIT_FAILURE);
30    }
31
32    /* Nous calculons le prix en solde : */
33    prix *= (1 - remise / 100.);
34
35    printf("L'article en solde vaut %g\n", prix);
36
37    /* Nous prenons une décision : */
```

```

38 if(argent >= prix) {
39     printf("J'achète !\n");
40 } else if(argent < 0.) {
41     printf("Vous ne seriez pas déjà à découvert ?\n");
42 } else {
43     printf("Il faudra encore économiser ...!\n");
44 }
45
46 exit(EXIT_SUCCESS);
47 }
```

Correction 30 (★★ Menu et addition).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int entree = 0;
6     int plat = 0;
7     int dessert = 0;
8     double addition = 0.;
9     printf("Voici nos entrées : \n 1. (1.50 €) Salade racontée.\n 2.
10    ↳ (2.50 €) Œufs embrouillés.\n 0. Non merci.\nVotre choix :
11    ↳ ");
12     scanf("%d", &entree);
13     printf("Voici nos plats : \n 1. (12 €) Pizza spéciale
14    ↳ informaticien.\n 2. (8 €) Pâtes spéciales étudiants.\n 0.
15    ↳ Non merci.\nVotre choix : ");
16     scanf("%d", &plat);
17     printf("Voici nos desserts : \n 1. (2.50 €) La cerise sur le
18    ↳ gâteau.\n 0. Non merci.\nVotre choix : ");
19     scanf("%d", &dessert);
20     if(entree == 0 && plat == 0 && dessert == 0) {
21         printf("Bien, passez une bonne journée.\n");
22         exit(EXIT_SUCCESS);
23     }
24     printf("Votre commande : \n");
25     switch(entree) {
26         case 1 : {
27             printf(" - (1.50 €) Salade racontée.\n");
28         }
29     }
30 }
```

```
23         addition += 1.5;
24     } break;
25     case 2 : {
26         printf(" - (2.5 €) Œufs embrouillés.\n");
27         addition += 2.5;
28     } break;
29     default : break;
30 }
31 switch(plat) {
32     case 1 : {
33         printf(" - (12 €) Pizza spéciale informaticien.\n");
34         addition += 12;
35     } break;
36     case 2 : {
37         printf(" - (8 €) Pâtes spéciales étudiants.\n");
38         addition += 8;
39     } break;
40     default : {
41         printf(" - Pas de plat...\n");
42     } break;
43 }
44 switch(dessert) {
45     case 1 : {
46         printf(" - (2.5 €) La cerise sur le gâteau.\n");
47         addition += 2.5;
48     } break;
49     default : break;
50 }
51 printf("Total hors taxes : %.2f €\n", addition);
52 addition *= 1.20;
53 printf("Total TTC %.2f € (TVA à 20%)\n", addition);
54
55 exit(EXIT_SUCCESS);
56 }
```

Correction 31 (★★★ Calculatrice).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```

4 int main() {
5     /* Nous obtenons le choix de l'utilisateur depuis un menu      */
6     /* proposé :                                                 */
7     int choix_menu;
8     printf("1 - Calculer\n2 - Quitter\n---\nVotre choix : ");
9     scanf("%d", &choix_menu);
10
11    /* Selon ce choix, nous y répondons : */
12    switch(choix_menu) {
13        case 1 :
14            break;
15        default :
16            printf("Le choix %d n'est pas proposé.\n", choix_menu);
17            /* En cas de choix invalide, nous continuons vers le      */
18            /* choix de s'arrêter :                                     */
19        case 2 :
20            /* Nous pouvons terminer le programme ici : */
21            printf("Bien, au revoir.\n");
22            exit(EXIT_SUCCESS);
23    }
24    /* Dans la suite de ce code, le choix était de calculer,      */
25    /* sinon le programme s'est terminé : ceci nous permet de      */
26    /* continuer dans une zone non indentée.                      */
27
28    /* Nous récupérons opérandes et opérateur comme dans une      */
29    /* calculatrice :                                              */
30    int first, second;
31    char op_symb;
32    printf(">>> ");
33    scanf("%d %c %d", &first, &op_symb, &second);
34
35    /* Nous affichons les valeurs récupérées puis le résultat      */
36    /* selon l'opérateur :                                         */
37    printf("%d %c %d = ", first, op_symb, second);
38    switch(op_symb) {
39
40        case '+' :
41            printf("%ld\n", (long)first + second);
42            break;
43
44        case '-' :

```

```
45     printf("%ld\n", (long)first - second);
46     break;
47
48     case '*':
49         printf("%ld\n", (long)first * second);
50         break;
51
52     case '/':
53         if(second == 0) {
54             printf("inf\n");
55             break;
56         }
57         printf("%lg (~ %d)\n", (double)first / second, first /
58             second);
59         break;
60
61     case '%':
62         if(second == 0) {
63             printf("NaN\n");
64             break;
65         }
66         printf("%d\n", first % second);
67         break;
68
69     default:
70         printf("?\n");
71     }
72     exit(EXIT_SUCCESS);
}
```

Correction 32 (★★ Racines d'un polynôme du second degré).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int main() {
6     double a, b, c;
7     printf("Soit ax^2 + bx + c une expression polynomiale,\nEntrez
8         les coefficients a, b et c :\n");
```

```

8   scanf("%lf %lf %lf", &a, &b, &c);
9
10  printf("Pour le polynôme P : x -> %lg.x^2 + %lg.x + %lg :\n", a,
11      b, c);
12
13  if(a != 0) {
14      double delta = b * b - 4 * a * c;
15
16      if(delta < 0) {
17          printf("Il n'y a pas de racine réelle.\n");
18
19      } else if(delta > 0) {
20          double x1 = (-b - sqrt(delta)) / (2 * a);
21          double x2 = (-b + sqrt(delta)) / (2 * a);
22
23          printf("Il y a deux racines réelles : %lg et %lg\n", x1,
24              x2);
25          printf("P(x) = %lg.x^2 + %lg.x + %lg = %lg.(x - %lg).(x -
26              %lg)\n", a, b, c, a, x1, x2);
27          printf("P(%lg) = %lg\n", x1, a * x1 * x1 + b * x1 + c);
28          printf("P(%lg) = %lg\n", x2, a * x2 * x2 + b * x2 + c);
29
30      } else {
31          double x0 = -b / (2 * a);
32
33          printf("Il y a une racine réelle : %lg\n", x0);
34          printf("P(x) = %lg.x^2 + %lg.x + %lg = %lg.(x - %lg)^2\n",
35              a, b, c, a, x0);
36          printf("P(%lg) = %lg\n", x0, a * x0 * x0 + b * x0 + c);
37      }
38  } else if(b != 0) {
39      double x0 = -c / b;
40
41      printf("Il y a une racine réelle : %lg\n", x0);
42      printf("P(x) = %lg.x^2 + %lg.x + %lg = %lg.(x - %lg)\n", a, b,
43          c, b, x0);
44      printf("P(%lg) = %lg\n", x0, a * x0 * x0 + b * x0 + c);
45
46  } else {
47      printf("P est une fonction constante égale à %lg\n", c);
48  }

```

```
44     exit(EXIT_SUCCESS);  
45 }
```

Correction 33 (★★★ Code obscurantiste).

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main() {  
5     int a, b;  
6  
7     printf("Entrez deux nombres : ");  
8     scanf("%d %d", &a, &b);  
9  
10    /* ! (a - b) vaut une valeur de vérité : */  
11    /* * ! (a - b) est vrai si (a - b) est faux */  
12    /* le faux est 0, donc (a - b) == 0 */  
13    /* d'où a == b */  
14    /* * ! (a - b) est faux si (a - b) est vrai */  
15    /* le vrai est toute valeur différente de 0 */  
16    /* donc, il suffit que a - b != 0, alors */  
17    /* a != b */  
18    /* ! (a - b) est donc vrai si a == b et faux si a != b, */  
19    /* alors ! (a - b) est équivalent à a == b */  
20    if(a == b) {  
21        printf("%d et %d sont égaux\n", a, b);  
22    } else {  
23        /* nous proposons l'ajout d'une variable qui offre plus de */  
24        /* sémantique sur le résultat obtenu. */  
25        /* Notons que si (a - b) est négatif, alors (a - b) ne */  
26        /* peut pas être représenté par la même valeur en signé */  
27        /* et non signé (bit de signe comme bit de poids fort) */  
28        /* (unsigned int)(a - b) donne donc une valeur différent */  
29        /* de (a - b). */  
30        /* (unsigned int)(a - b) ne tient pas sur la plage signée */  
31        /* des int (4 octets) mais peut se représenter sur la */  
32        /* plage signée des long (8 octets). */  
33        /* Par conséquent si (a - b) est positif, sa valeur reste */  
34        /* inchangée par la transformation */
```

```

36  /* (long)(unsigned int)(a - b), mais deviendra un résultat */
37  /* positif si (a - b) est négatif. */
38  /* ((long)(unsigned int)(a - b) == a - b) est donc vraie */
39  /* si a - b >= 0 d'où a >= b et fausse si a - b < 0 . */
40  /* dans le cas vrai, l'expression ternaire vaut b et dans */
41  /* le cas vrai vaut a. */
42  /* une possibilité est donc de proposer la vérification de */
43  /* (a < b) pour donner a si vrai et b si faux. */
44  /* Ceci offre plus de lisibilité et pour ajouter du sens à */
45  /* cette expression, c'est en réalité le calcul du minimum */
46  /* des deux valeurs. */
47  int minimum = (a < b) ? a : b;
48  printf("Entre %d et %d, le plus petit est %d\n", a, b,
49      → minimum);
50
51 }
52 exit(EXIT_SUCCESS);
}

```

Correction 34 (★★★ Validation d'un mot de passe).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int first_pass = 0, second_pass = 0;
6
7     /* Pour tester le code, il est possible d'ajouter la lecture */
8     /* des valeurs à l'exécution : */
9     printf("Entrez les deux codes secrets : ");
10    scanf("%d %d", &first_pass, &second_pass);
11
12    /* Les combinaisons possibles sont : */
13    /* first_pass / second_pass */
14    /* -----+----- */
15    /*        42 | 1337      */
16    /*        1337 | 42      */
17    /*        1235 | 1235      */
18
19    /* Dans le cas de (1337, 42), first_pass > second_pass */

```

```
20 if(first_pass > second_pass) {
21     int tmp = first_pass;
22     first_pass = second_pass;
23     second_pass = tmp;
24 }
25 /* Après l'échange, les combinaisons sont réduites à */
26 /* first_pass / second_pass */
27 /* -----+----- */
28 /*      42 | 1337      */
29 /*      1235 | 1235      */

30
31 /* La condition de l'accès refusé n'est pas logiquement */
32 /* triviale. Une proposition est de considérer :
33 /* acces_refuse = ! acces_autorise
34 /* Pour acces_autorise, ceci est vrai si une des
35 /* combinaisons est donnée :
36 /* acces_autorise = (combinaison 42, 1337) ||
37 /* (combinaison 1235, 1235)
38 /* La vérification d'une combinaison demande que les deux
39 /* mots de passe soient vérifiés :
40 /* (combinaison a, b) = (first_pass == a &&
41 /* second_pass == b)
42 /* À partir de cette construction, il est possible de
43 /* construire la condition d'accès refusé comme étant la
44 /* suivante :
45 if( !(first_pass == 42 && second_pass == 1337)
46   || (first_pass == 1235 && second_pass == 1235)) {
47
48     printf("Accès refusé.\n");
49     exit(EXIT_SUCCESS);
50 }
51
52     printf("Bienvenue !\n");
53     exit(EXIT_SUCCESS);
54 }
```

Correction 35 (★★★ Calcul de l'impôt sur le revenu).

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```

3 int main() {
4     const int finT1 = 10084;
5     const int finT2 = 25710;
6     const int finT3 = 73516;
7     const int finT4 = 158122;
8     const float tauxT1 = 0.00;
9     const float tauxT2 = 0.11;
10    const float tauxT3 = 0.30;
11    const float tauxT4 = 0.41;
12    const float tauxT5 = 0.45;
13
14
15    unsigned char couple = 0;
16    unsigned int enfants = 0;
17    float parts = 0;
18    unsigned long revenusFoyer = 0;
19    unsigned long revenusConjoint = 0;
20    unsigned long quotientFamillial = 0;
21    unsigned long impot = 0;
22
23    printf("Quels sont vos revenus annuels nets personnels ?\n");
24    scanf("%lu", &revenusFoyer);
25    printf("Êtes-vous marié ou pacsé ? (Y ou n)\n");
26    scanf(" %c", &couple);
27    couple = (couple == 'Y' || couple == 'y' || couple == 'O' ||
28              couple == 'o');
29    if(couple) {
30        printf("Quels sont les revenus annuels nets de votre conjoint
31              ?\n");
32        scanf("%lu", &revenusConjoint);
33        revenusFoyer += revenusConjoint;
34    }
35    printf("Combien avez-vous d'enfants ?\n");
36    scanf("%u", &enfants);
37    parts = 1 + 0.5 * enfants;
38    if(couple) {
39        parts += 1;
40    } else if(enfants > 0) {
41        parts += 0.5;
42    }
43    quotientFamillial = revenusFoyer / parts;

```

```
42
43     if(quotientFamillial < finT1) {
44         impot = quotientFamillial * tauxT1;
45     } else {
46         impot = finT1 * tauxT1;
47         if(quotientFamillial < finT2) {
48             impot += (quotientFamillial - finT1) * tauxT2;
49         } else {
50             impot += (finT2 - finT1) * tauxT2;
51             if(quotientFamillial < finT3) {
52                 impot += (quotientFamillial - finT2) * tauxT3;
53             } else {
54                 impot += (finT3 - finT2) * tauxT3;
55                 if(quotientFamillial < finT4) {
56                     impot += (quotientFamillial - finT3) * tauxT4;
57                 } else {
58                     impot += (finT4 - finT3) * tauxT4;
59                     impot += (quotientFamillial - finT4) * tauxT5;
60                 }
61             }
62         }
63     }
64
65     printf("Ceci vous fait donc un impôt de %lu x %g = %lu €\n",
66           ↳ impot, parts, (unsigned long)(impot * parts));
67
68     exit(EXIT_SUCCESS);
}
```

Correction 36 (∞ Lancé de rayon sur sphères).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5 #include <SDL/SDL.h>
6 #include <SDL/SDL_gfxPrimitives.h>
7
8 /* Paramètres de la fenêtre : */
9 const int largeur = 800;
```

```

10 const int hauteur = 600;
11 const char * titre = "ESGI spheres";
12
13 const int spheres_count = 8;
14 double temps = 0;
15 const int pixel_size = 2;
16
17 double camera_position_x = 0.;
18 double camera_position_y = 0.;
19 double camera_position_z = -5.;

20
21 /* Implicite :
22    double camera_direction_x = 0.;
23    double camera_direction_y = 0. ;
24    double camera_direction_z = 1. ;
25 */
26
27 double sphere_intersection(double start_x, double start_y, double
→ start_z, double direction_x, double direction_y, double
→ direction_z, double center_x, double center_y, double
→ center_z, double radius) {
28    double direction_norm_square = pow(direction_x, 2.) +
→    pow(direction_y, 2.) + pow(direction_z, 2.);
29    double positions_norm_square = pow(start_x - center_x, 2.) +
→    pow(start_y - center_y, 2.) + pow(start_z - center_z, 2.);
30    double a = direction_norm_square;
31    double b = 2. * (direction_x * (start_x - center_x) +
→    direction_y * (start_y - center_y) + direction_z * (start_z
→    - center_z));
32    double c = positions_norm_square - radius * radius;
33    double d = b * b - 4 * a * c;
34    if(d < 0) {
35        return -1e100;
36    }
37    double x1 = (-b - sqrt(d)) / (2 * a);
38    double x2 = (-b + sqrt(d)) / (2 * a);
39    if(x1 > 0 && x2 > 0) {
40        return fmin(x1, x2);
41    } else if(x1 > 0) {
42        return x1;
43    } else if(x2 > 0) {

```

```
44     return x2;
45 }
46 return -1e100;
47 }

48
49 const unsigned int R_CHANNEL = 0x01000000;
50 const unsigned int G_CHANNEL = 0x00010000;
51 const unsigned int B_CHANNEL = 0x00000100;
52 const unsigned int A_CHANNEL = 0x00000001;

53
54 double intensity(double x, double y, double z, double center_x,
55   → double center_y, double center_z) {
56     x -= center_x;
57     y -= center_y;
58     z -= center_z;
59     double norm = sqrt(x * x + y * y + z * z);
60     if(norm < 1e-9) {
61       return 0.5;
62     }
63     x /= norm;
64     y /= norm;
65     z /= norm;
66     double lx = 1.;
67     double ly = -1.;
68     double lz = -1.;
69     norm = sqrt(lx * lx + ly * ly + lz * lz);
70     lx /= norm;
71     ly /= norm;
72     lz /= norm;
73     return fmax(pow(0.5 * (x * lx + y * ly + z * lz) + 0.5, 0.75),
74      → 0.25);
75 }
76
77 int nearest(int first_sphere, double first_offset, int
78   → second_sphere, double second_offset) {
79     if(first_offset > 0 && second_offset > 0) {
80       return (first_offset < second_offset) ? first_sphere :
81         → second_sphere;
82     } else if(first_offset > 0) {
83       return first_sphere;
84     } else if(second_offset > 0) {
```

```

81     return second_sphere;
82 }
83 return (first_offset < second_offset) ? first_sphere :
84     → second_sphere;
85 }
86
87 unsigned int sphere_color(int id) {
88     unsigned char r, g, b, a;
89     a = 255;
90     r = (id * 1337 + 127) % 256;
91     g = (id * 133712 + 127) % 256;
92     b = (id * 13371234 + 127) % 256;
93     return r * R_CHANNEL + g * G_CHANNEL + b * B_CHANNEL + a *
94         → A_CHANNEL;
95 }
96
97 double sphere_x(int id) {
98     return sin(id * 1274 + 0.2 * temps);
99 }
100
101 double sphere_y(int id) {
102     return sin(id * 127478 + (0.3 + id * 0.1) * temps);
103 }
104
105 double sphere_z(int id) {
106     return sin(id * 8899 + 0.5 * temps);
107 }
108
109 double sphere_r(int id) {
110     return 0.375 + 0.25 * sin(id * 855) + 0.1 * sin(temps);
111 }
112
113 unsigned int pixel_color(int x, int y) {
114     unsigned char r, g, b, a;
115     double offset_x = x / (largeur / 2.) - 1.;
116     double offset_y = y / (hauteur / 2.) - 1.;
117     offset_x *= ((float)largeur / hauteur);
118
119     int nearest_sphere;
120     double nearest_t;
121     int i;

```

```
120 double t;
121 for(i = 0; i < spheres_count; ++i) {
122     t = sphere_intersection(camera_position_x + offset_x,
123                             ↵ camera_position_y + offset_y, camera_position_z, 0., 0.,
124                             ↵ 1., sphere_x(i), sphere_y(i), sphere_z(i), sphere_r(i));
125     if(i == 0) {
126         nearest_sphere = i;
127         nearest_t = t;
128     } else {
129         nearest_sphere = nearest(nearest_sphere, nearest_t, i, t);
130         if(nearest_sphere == i) {
131             nearest_t = t;
132         }
133     }
134     if(nearest_t < 0.) {
135         return 0;
136     }
137     unsigned int color = sphere_color(nearest_sphere);
138     r = (color / R_CHANNEL) % 256;
139     g = (color / G_CHANNEL) % 256;
140     b = (color / B_CHANNEL) % 256;
141     double value = intensity(camera_position_x + offset_x,
142                               ↵ camera_position_y + offset_y, camera_position_z + nearest_t,
143                               ↵ sphere_x(nearest_sphere), sphere_y(nearest_sphere),
144                               ↵ sphere_z(nearest_sphere));
145     a = 255;
146     r = value * r;
147     g = value * g;
148     b = value * b;
149     return r * R_CHANNEL + g * G_CHANNEL + b * B_CHANNEL + a *
150           ↵ A_CHANNEL;
151 }
152 void affichage(SDL_Surface * ecran) {
153     /* Remplissage de l'écran par un gris foncé uniforme : */
154     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
155                                         ↵ 102));
156     /* Affichage du joueur : */
157     int x, y;
```

```

154     unsigned int color;
155     unsigned char r, g, b, a;
156     SDL_Rect pixel;
157     for(x = 0; x < largeur; x += pixel_size) {
158         for(y = 0; y < hauteur; y += pixel_size) {
159             color = pixel_color(x + pixel_size / 2., y + pixel_size /
160             ↳ 2.);
161             r = (color / R_CHANNEL) % 256;
162             g = (color / G_CHANNEL) % 256;
163             b = (color / B_CHANNEL) % 256;
164             a = (color / A_CHANNEL) % 256;
165             if(a < 127) continue;
166             pixel.x = x;
167             pixel.y = y;
168             pixel.w = pixel_size;
169             pixel.h = pixel_size;
170             SDL_FillRect(ecran, &pixel, SDL_MapRGB(ecran->format, r, g,
171             ↳ b));
172         }
173     }
174
int main() {
    srand(time(NULL));
    /* Création d'une fenêtre SDL : */
176    if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
        fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
        exit(EXIT_FAILURE);
    }
    SDL_Surface * ecran = NULL;
    if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
182    ↳ | SDL_DOUBLEBUF)) == NULL) {
        fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
        ↳ SDL_GetError());
        SDL_Quit();
        exit(EXIT_FAILURE);
    }
    SDL_WM_SetCaption(titre, NULL);
188
    int active = 1;
    SDL_Event event;

```

```
191 int FPS_delay = 1000 / 60;
192 int delay;
193
194 while(active) {
195
196     temps = SDL_GetTicks() / 1000.;
197     delay = SDL_GetTicks();
198     affichage(ecran);
199     SDL_Flip(ecran);
200
201     while(SDL_PollEvent(&event)) {
202
203         switch(event.type) {
204             /* Utilisateur clique sur la croix de la fenêtre : */
205             case SDL_QUIT : {
206                 active = 0;
207             } break;
208
209             /* Utilisateur enfonce une touche du clavier : */
210             case SDL_KEYDOWN : {
211                 switch(event.key.keysym.sym) {
212                     /* Touche Echap : */
213                     case SDLK_ESCAPE : {
214                         active = 0;
215                     } break;
216                 }
217             } break;
218         }
219     }
220
221     delay = FPS_delay - (SDL_GetTicks() - delay);
222     if(delay > 0) {
223         SDL_Delay(delay);
224     }
225 }
226
227     SDL_FreeSurface(ecran);
228     SDL_Quit();
229     exit(EXIT_SUCCESS);
230 }
```

Correction 37 (∞ Saut d'un personnage et suivi par caméra).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5 #include <SDL/SDL.h>
6 #include <SDL/SDL_gfxPrimitives.h>
7
8 /* Paramètres de la fenêtre : */
9 const int largeur = 800;
10 const int hauteur = 800;
11 const char * titre = "ESGI jump";
12
13 /* Coordonnées du joueur (position) : */
14 float px, py;
15 /* Vitesse du joueur (déplacement) : */
16 float vx, vy;
17 /* Accélération du joueur (saut, eau) : */
18 float ax, ay;
19
20 /* Coordonnées de l'objectif : */
21 float ox, oy;
22 /* Coordonnées de l'adversaire : */
23 float ex, ey;
24
25 /* Position de la caméra et échelle de la vue : */
26 float cx, cy, cz;
27
28 float distance(float first_x, float first_y, float second_x, float
→ second_y) {
29     return sqrt((first_x - second_x) * (first_x - second_x) +
→ (first_y - second_y) * (first_y - second_y));
30 }
31
32 float hauteur_terrain(float x) {
33     return 0.5 * (200 * sin(x / 400.) + 100 * sin(x / 117.) + 25 *
→ sin(x / 51.) + 10 * sin(x / 12.) + 10 * sin((x + 10) /
→ 13.));
34 }
35

```

```
36 void placer_objectif() {
37     int distance = (rand() % 1000) + 200;
38     if(rand() % 2 == 0) {
39         distance *= -1;
40     }
41     ox = px + distance;
42     oy = hauteur_terrain(ox) - 200;
43 }
44
45 void placer_adversaire() {
46     int distance = 5000;
47     if(rand() % 2 == 0) {
48         distance *= -1;
49     }
50     ex = px + distance;
51     ey = hauteur_terrain(ex);
52 }
53
54 int ecran_depuis_camera_x(float x) {
55     return ((x - cx) * largeur / 2) / cz + largeur / 2;
56 }
57
58 int ecran_depuis_camera_y(float y) {
59     return ((y - cy) * hauteur / 2) / cz + hauteur / 2;
60 }
61
62 float camera_depuis_ecran_x(int x) {
63     return 2. * ((x - largeur / 2) * cz) / largeur + cx;
64 }
65
66 float camera_depuis_ecran_y(int y) {
67     return 2. * ((y - hauteur / 2) * cz) / hauteur + cy;
68 }
69
70 float ecran_depuis_camera_z(int z) {
71     return (z * largeur / 2.) / cz;
72 }
73
74 float camera_depuis_ecran_z(int z) {
75     return (2 * z * cz) / largeur;
76 }
```

```

77
78 void affichage(SDL_Surface * ecran) {
79     /* Remplissage de l'écran par un gris foncé uniforme : */
80     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
81     → 102));
82     /* Affichage du joueur : */
83     filledCircleRGBA(ecran, ecran_depuis_camera_x(px),
84     → ecran_depuis_camera_y(py), ecran_depuis_camera_z(25), 51,
85     → 204, 51, 255);
86
87     filledCircleRGBA(ecran, ecran_depuis_camera_x(ox),
88     → ecran_depuis_camera_y(oy), ecran_depuis_camera_z(25), 204,
89     → 204, 51, 255);
90
91     filledCircleRGBA(ecran, ecran_depuis_camera_x(ex),
92     → ecran_depuis_camera_y(ey), ecran_depuis_camera_z(50), 204,
93     → 51, 51, 255);
94
95     int x, y, z;
96     for(x = 0; x < largeur; ++x) {
97         y =
98             → ecran_depuis_camera_y(hauteur_terrain(camera_depuis_ecran_x(x)));
99         z = ecran_depuis_camera_y(0);
100        boxRGBA(ecran, x, y, x + 1, z, 51, 51, 204, 127);
101        boxRGBA(ecran, x, y, x + 1, hauteur, 102, 51, 51, 255);
102        boxRGBA(ecran, x, y, x + 1, y + ecran_depuis_camera_z(5), 51,
103             → 102, 51, 255);
104    }
105 }
106
107 int calculs(int score) {
108     ay += (py > 0) ? -1 : 10;
109     vx += ax;
110     vy += ay;
111     px += vx;
112     py += vy;
113     ax = 0;
114     ay = 0;
115     int ht = hauteur_terrain(px);
116     if(py > ht) {
117         py = ht;

```

```
109     vy = 0;
110 }
111
112 if(distance(px, py, ox, oy) < 50) {
113     score += 1;
114     placer_objectif();
115 }
116
117 if(distance(px, py, ex, ey) < 50) {
118     score -= 10;
119     if(score < 0) {
120         score = 0;
121     }
122     placer_adversaire();
123 }
124
125 int av = (ey > 0) ? 2 : 5;
126 if(px < ex) {
127     ex -= av;
128 } else {
129     ex += av;
130 }
131 ey = hauteur_terrain(ex);
132
133 return score;
134 }
135
136 void action_jump() {
137     ay -= (py > 0) ? 40 : 80;
138 }
139
140 void action_droite() {
141     int move = (py > 0) ? 8 : 20;
142     vx = move;
143 }
144
145 void action_gauche() {
146     int move = (py > 0) ? 8 : 20;
147     vx = -move;
148 }
```

```
150 void action_sans_direction() {
151     vx = 0;
152 }
153
154 void deplacer_camera() {
155     float d = distance(cx, cy, px, py);
156     if(d > cz / 2.) {
157         cx -= 0.125 * d * (cx - px) / (2. * cz);
158         cy -= 0.125 * d * (cy - py) / (2. * cz);
159     }
160 }
161
162 int main() {
163     srand(time(NULL));
164     /* Création d'une fenêtre SDL : */
165     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
166         fprintf(stderr, "Error in SDL_Init : %s\n",
167                 SDL_GetError());
168         exit(EXIT_FAILURE);
169     }
170     SDL_Surface * ecran = NULL;
171     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
172         | SDL_DOUBLEBUF)) == NULL) {
173         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
174                 SDL_GetError());
175         SDL_Quit();
176         exit(EXIT_FAILURE);
177     }
178     SDL_WM_SetCaption(titre, NULL);

179     /* Placement du joueur au centre de la fenêtre : */
180     px = 0;
181     py = 0;
182     vx = 0;
183     vy = 0;
184     ax = 0;
185     ay = 0;

186     cx = 0;
187     cy = 0;
188     cz = 500;
```

```
189 int active = 1;
190 SDL_Event event;
191 int moving_right = 0;
192 int moving_left = 0;
193 int jump = 0;
194
195 int score = 0;
196 unsigned int temps;
197 char display[100];
198
199 placer_objectif();
200
201 while(active) {
202
203     temps = SDL_GetTicks();
204     affichage(ecran);
205     sprintf(display, "Score : %d", score);
206     stringRGBA(ecran, 5, 5, display, 255, 255, 255, 255);
207     sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) /
208             → 60, (temps / 1000) % 60, (temps % 1000) / 10);
209     stringRGBA(ecran, 5, 25, display, 255, 255, 255, 255);
210     SDL_Flip(ecran);
211
212     while(SDL_PollEvent(&event)) {
213
214         switch(event.type) {
215             /* Utilisateur clique sur la croix de la fenêtre : */
216             case SDL_QUIT : {
217                 active = 0;
218                 } break;
219
220             /* Utilisateur enfonce une touche du clavier : */
221             case SDL_KEYDOWN : {
222                 switch(event.key.keysym.sym) {
223                     /* Touche Echap : */
224                     case SDLK_ESCAPE : {
225                         active = 0;
226                         } break;
227
228                     case SDLK_z :
229                     case SDLK_UP : {
```

```
229         if(! jump) {
230             action_jump();
231         }
232         jump = 1;
233     } break;
234
235     case SDLK_d :
236     case SDLK_RIGHT : {
237         moving_right = 1;
238     } break;
239
240     case SDLK_q :
241     case SDLK_LEFT : {
242         moving_left = 1;
243     } break;
244 }
245 } break;
246
247 case SDL_KEYUP : {
248     switch(event.key.keysym.sym) {
249         case SDLK_z :
250         case SDLK_UP : {
251             jump = 0;
252         } break;
253
254         case SDLK_d :
255         case SDLK_RIGHT : {
256             moving_right = 0;
257         } break;
258
259         case SDLK_q :
260         case SDLK_LEFT : {
261             moving_left = 0;
262         } break;
263     }
264 } break;
265
266 case SDL_MOUSEBUTTONDOWN : {
267     switch(event.button.button) {
268         case SDL_BUTTON_WHEELUP : {
269             cz *= 0.8;
```

```
270         if(cz < 1) {
271             cz = 1;
272         }
273     } break;
274
275     case SDL_BUTTON_WHEELDOWN : {
276         cz /= 0.8;
277         if(cz > 10000) {
278             cz = 10000;
279         }
280     } break;
281 }
282 } break;
283 }
284 }
285
286 if(moving_right && moving_left) {
287     action_sans_direction();
288 } else if(moving_right) {
289     action_droite();
290 } else if(moving_left) {
291     action_gauche();
292 } else {
293     action_sans_direction();
294 }
295 score = calculs(score);
296
297 placer_camera();
298
299 if(score >= 10) {
300     affichage(ecran);
301     sprintf(display, "Score : %d", score);
302     stringRGBA(ecran, 5, 5, display, 255, 255, 255);
303     sprintf(display, "Temps : %u:%02u:%02u s", (temps / 1000) /
304             60, (temps / 1000) % 60, (temps % 1000) / 10);
305     stringRGBA(ecran, 5, 25, display, 255, 255, 255);
306     stringRGBA(ecran, largeur / 2 - 10, hauteur / 2, "BRAVO !",
307                 255, 255, 255);
308     SDL_Flip(ecran);
309     SDL_Delay(3000);
310     active = 0;
```

```
309 }
310     SDL_Delay(1000 / 60);
311 }
312
313     SDL_FreeSurface(ecran);
314     SDL_Quit();
315     exit(EXIT_SUCCESS);
316 }
317 }
```

A.5 Boucles

A.5.1 Entraînement

Correction 38 (★★ Liste des diviseurs).

```
1  /* Première proposition (traitement du premier hors boucle) : */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main() {
6      int nombre;
7      int i;
8      /* Nous demandons la saisie d'un entier à l'utilisateur : */
9      printf("Entrez un entier : ");
10     scanf("%d", &nombre);
11     printf("Liste des diviseurs de %d :\n", nombre);
12
13     /* 1 divisera toujours notre nombre. */
14     printf("1");
15     /* On itère par l'entier i sur les entiers de 2 à nombre */
16     for(i = 2; i <= nombre; ++i) {
17         /* si i divise nombre, on l'affiche */
18         if(nombre % i == 0) {
19             printf(", %d", i);
20         }
21     }
22     printf("\n");
23
24     exit(EXIT_SUCCESS);
25 }
```

```
1  /* Seconde proposition (tout traité dans la boucle) : */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main() {
6      int nombre;
7      int i;
8      /* Nous demandons la saisie d'un entier à l'utilisateur : */
```

```

9  printf("Entrez un entier : ");
10 scanf("%d", &nombre);
11 printf("Liste des diviseurs de %d :\n", nombre);
12
13 for(i = 1; i <= nombre; ++i) {
14     if(nombre % i == 0) {
15         if(i > 1) {
16             printf(", ");
17         }
18         printf("%d", i);
19     }
20 }
21 printf("\n");
22
23 exit(EXIT_SUCCESS);
24 }
```

Correction 39 (★★★ Force brute).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Nous récupérons les informations fournies par Oscar : */
6     unsigned int k = 2015201261;
7     unsigned int p = 4285404239;
8     /* Nous sauvegarderons la clé secrète d'Oscar dans s : */
9     unsigned int s = 1;
10
11    /* Nous recherchons itérativement la solution à l'équation */
12    /* qui validerait la clé secrète : */
13    while(((unsigned long)k * s) % p != 1) {
14        ++s;
15    }
16
17    /* Nous affichons l'équation : */
18    printf("%u * %u = %lu [%u]\n", s, k, ((unsigned long)k * s) % p,
19           p);
20
21    /* Nous vérifions que la clé fonctionne pour l'exemple donné */
```

```
21  /* dans un exercice précédent : */  
22  if(((unsigned long)s * 0xfee1900d) % p) == 0x5c003212) {  
23  
24      printf("Nous avons la clé secrète d'Oscar !\nCette clé vaut  
25          → %u (0x%x)", s, s);  
26  } else {  
27  
28      printf("Visiblement, ceci n'a pas fonctionné...\n");  
29  }  
30  
31      exit(EXIT_SUCCESS);  
32 }
```

Correction 40 (★★★ Affichage en binaire).

```
1  /* Version petits entiers (maximum : environ 4 000 000) */  
2  #include <stdio.h>  
3  #include <stdlib.h>  
4  
5  int main() {  
6      unsigned int nombreUser;  
7      unsigned int nombre;  
8  
9      /* base10 permet de représenter les puissances de 2 */  
10     unsigned long base10 = 1;  
11     /* binaireView permet de sauvegarder la représentation */  
12     /* binaire sous une représentation octale : */  
13     unsigned long binaireView = 0;  
14  
15     /* Nous demandons la saisie d'un entier à l'utilisateur : */  
16     printf("Entrez un nombre : ");  
17     scanf("%u", &nombreUser);  
18     nombre = nombreUser;  
19  
20     /* Tant que le nombre sur lequel nous travaillons est */  
21     /* différent de 0 */  
22     while(nombre) {  
23         /* Nous ajoutons un 1 si son reste modulo 2 fait 1 */  
24         if(nombre % 2 == 1) {
```

```

25     binaireView += base10;
26 }
27 /* Nous passons au chiffre suivant pour la puissance de 2 */
28 /* représenté en base octale */
29 base10 *= 010;
30 /* Nous divisons le nombre par 2 pour passer au chiffre */
31 /* binaire suivant */
32 nombre /= 2;
33 }
34
35 /* Nous affichons le résultat en base octale ce qui donne la */
36 /* représentation binaire pour des petits nombres : */
37 printf("%u = (%lo_2\n", nombreUser, binaireView);
38 exit(EXIT_SUCCESS);
39 }
```

```

1 /* Version grands entiers */
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     unsigned long nombreUser;
7     unsigned long nombre;
8
9     /* Nous construisons la plus grande puissance de 2 par */
10    /* l'hexadécimal. Nous avons besoin de construire le plus */
11    /* grand nombre en binaire commençant par 1 : */
12    /* 1000 0000 ... 0000 */
13     unsigned long base2 = 0x8000000000000000;
14
15     /* Nous récupérons la saisie de l'utilisateur : */
16     printf("Entrez un nombre : ");
17     scanf("%lu", &nombreUser);
18     nombre = nombreUser;
19
20     /* Nous recherchons la plus grande puissance de 2 que l'on */
21     /* pourrait soustraire au nombre de l'utilisateur. */
22     /* (Ceci permet d'éviter l'affichage de 0 inutiles) */
23     while(base2 > nombre) {
24         base2 /= 2;
```

```
25 }
26
27 /* Nous commençons l'affichage : */
28 printf("%lu = ", nombreUser);
29 /* Tant que nous avons une puissance de 2 : */
30 while(base2 > 0) {
31     /* Si la puissance peut se soustraire au nombre */
32     if(base2 <= nombre) {
33         /* Nous affichons le 1 présent dans la représentation */
34         /* binaire */
35         printf("1");
36         nombre -= base2;
37     } else {
38         /* Sinon la puissance est absente de la représentation */
39         /* binaire */
40         printf("0");
41     }
42     /* Nous passons à la puissance suivante */
43     base2 /= 2;
44 }
45 printf(")_2\n");
46 exit(EXIT_SUCCESS);
47 }
```

Correction 41 (★★★ PGCD).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int first, second;
6     int a, b;
7     int tmp;
8     int q, r;
9
10    /* Nous demandons deux entiers à l'utilisateur */
11    printf("Entrez deux entiers : ");
12    scanf("%d %d", &first, &second);
13    /* Nous copions ces valeurs dans des variables de travail : */
14    a = first;
```

```

15   b = second;
16
17   /* Si la première est plus petite, nous les échangeons : */
18   if(a < b) {
19     tmp = a;
20     a = b;
21     b = tmp;
22   }
23
24   /* Tant que le second est différent de 0 */
25   /* (le second sera le reste précédent) */
26   while(b != 0) {
27     /* Nous calculons quotient et reste par la division           */
28     /* euclidienne :                                              */
29     q = a / b;
30     r = a % b;
31     /* Nous les affichons : */
32     printf("%d = %d * %d + %d\n", a, b, q, r);
33     /* Nous préparons l'étape suivante : */
34     a = b;
35     b = r;
36   }
37
38   /* Le résultat est le dernier reste non nul : sauvegardé      */
39   /* dans a                                                       */
40   printf("pgcd(%d, %d) = %d\n", first, second, a);
41
42   exit(EXIT_SUCCESS);
43 }
```

Correction 42 (★ Exponentiation d'un entier).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 int main() {
6   int n;
7   unsigned long value;
8   unsigned long result = 1;
```

```
9  printf("Entrez un entier : ");
10 scanf("%lu", &value);
11 printf("Entrez un exposant : ");
12 scanf("%d", &n);
13 if(n * log(value) >= (8 * sizeof(long) - 0.1) * log(2)) {
14     printf("Cette exponentiation n'est pas assez triviale pour ce
15         → programme (le résultat doit être sous 2 puissance 64)\n");
16     exit(EXIT_FAILURE);
17 } else if(n < 0) {
18     result = 0;
19 }
20 printf("%lu puissance %d = ", value, n);
21 for(; n > 0; --n) {
22     result *= value;
23 }
24 printf("%lu\n", result);
25 exit(EXIT_SUCCESS);
}
```

Correction 43 (★★★ Jeu du plus ou moins).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 int main() {
6     /* srand permet d'initialiser le générateur pseudo aléatoire */
7     /* à une graine donnée. */
8     /* Pour avoir un aléatoire variant d'une exécution à           */
9     /* l'autre, nous l'initialisons par le nombre de secondes       */
10    /* écoulées depuis le 1er Janvier 1970.                         */
11    srand(time(NULL));
12
13    const int max = 1000;
14    /* rand renvoie une valeur aléatoire sur un int.               */
15    /* En effectuant un modulo, nous récupérons une valeur          */
16    /* aléatoire tronquée de celle-ci vivant entre 0 et max :      */
17    int nombre = rand() % (max + 1);
18
19    int user;
```

```

20   printf("Nous avons choisi un nombre entre 0 et %d\n", max);
21   do {
22     /* Nous récupérons la saisie de l'utilisateur : */
23     printf("A quel nombre pensez-vous ? ");
24     scanf("%d", &user);
25
26     /* Nous lui indiquons le positionnement de son nombre par */
27     /* rapport au nombre caché : */
28     if(user < nombre) {
29       printf("Trop petit.\n");
30     } else if(user > nombre) {
31       printf("Trop grand.\n");
32     }
33     /* Nous continuerais tant que le nombre de l'utilisateur */
34     /* n'est pas le même que le nombre caché. */
35   } while (user != nombre);
36
37   printf("Bien joué, le nombre était en effet %d\n", nombre);
38
39   exit(EXIT_SUCCESS);
40 }

```

Correction 44 (★★★ Implémentation racine carré).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef COMPARE
4 #include <math.h>
5 #endif
6 #include <time.h>
7
8 int main() {
9   clock_t start, end;
10  double x;
11  double target;
12  printf("Entrez un réel : ");
13  scanf("%lf", &target);
14  start = clock();
15  x = target / 2.;
```

```
16 int i;
17 for(i = 0; i < 200; ++i) {
18     x = (x * x + target) / (2 * x);
19 }
20 end = clock();
21 printf("La racine carré de %g est %.16f / %g (en %.6f s)\n",
22       → target, x, x, (double)(end - start) / CLOCKS_PER_SEC);
23 #ifdef COMPARE
24     start = clock();
25     x = sqrt(target);
26     end = clock();
27     printf("La racine carré de %g est %.16f / %g (en %.6f s)
28           → (d'après math)\n", target, x, x, (double)(end - start) /
29           → CLOCKS_PER_SEC);
29 #endif
30     exit(EXIT_SUCCESS);
31 }
```

Correction 45 (∞ Dessin de fractales).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5 #include <SDL/SDL.h>
6 #include <SDL/SDL_gfxPrimitives.h>
7
8 /* Paramètres de la fenêtre : */
9 const int largeur = 800;
10 const int hauteur = 800;
11 const char * titre = "ESGI draw";
12 const int functions_count = 8;
13 int resolution = 1;
14
15 double jx = 0, jy = 0;
16 int max_iter = 25;
17 int newton_n = 3;
18
19 /* Position de la caméra et échelle de la vue : */
20 float cx, cy, cz;
```

```
21 int current_function = 0;
22
23 int (* fonction(int id))(double x, double y);
24
25 const char * fonction_name(int id);
26
27 int ecran_depuis_camera_x(float x) {
28     return ((x - cx) * largeur / 2) / cz + largeur / 2;
29 }
30
31 int ecran_depuis_camera_y(float y) {
32     return ((y - cy) * hauteur / 2) / cz + hauteur / 2;
33 }
34
35 float ecran_depuis_camera_z(int z) {
36     return (z * largeur / 2.) / cz;
37 }
38
39 float camera_depuis_ecran_x(int x) {
40     return 2. * ((x - largeur / 2) * cz) / largeur + cx;
41 }
42
43 float camera_depuis_ecran_y(int y) {
44     return 2. * ((y - hauteur / 2) * cz) / hauteur + cy;
45 }
46
47 float camera_depuis_ecran_z(int z) {
48     return (2 * z * cz) / largeur;
49 }
50
51 const unsigned int R_CHANNEL = 0x01000000;
52 const unsigned int G_CHANNEL = 0x00010000;
53 const unsigned int B_CHANNEL = 0x00000100;
54 const unsigned int A_CHANNEL = 0x00000001;
55
56 int implicit_distance(double iso) {
57     double v = 0.5 * cos(iso * M_PI * 2.) + 0.5;
58     if(iso < 0) {
59         v = 0.5 + 0.5 * v;
60     }
61 }
```

```
62     unsigned char r, g, b, a;
63     a = 255;
64     if(fabs(v) > 1 - 1e-2 && fabs(iso) < 0.1) {
65         r = 0;
66         g = 255;
67         b = 0;
68     } else if(fabs(v) > 1 - 1e-2) {
69         r = 127;
70         g = 127;
71         b = 127;
72     } else if(iso < 0) {
73         r = 255;
74         g = 0;
75         b = 0;
76     } else {
77         r = 0;
78         g = 0;
79         b = 255;
80     }
81     r = 0.75 * v * r + 63;
82     g = 0.75 * v * g + 63;
83     b = 0.75 * v * b + 63;
84     return r * R_CHANNEL + g * G_CHANNEL + b * B_CHANNEL + a *
85     → A_CHANNEL;
86 }
87
88 int simple_distance(double iso) {
89     double v = 1;
90     unsigned char r, g, b, a;
91     a = 255;
92     r = 255;
93     g = 255;
94     b = 255;
95     if(iso > 0) v = 0;
96     r = 0.75 * v * r + 63;
97     g = 0.75 * v * g + 63;
98     b = 0.75 * v * b + 63;
99     return r * R_CHANNEL + g * G_CHANNEL + b * B_CHANNEL + a *
100    → A_CHANNEL;
}
```

```

101 int factual_color(double coeff) {
102     double v = 1;
103     unsigned char r, g, b, a;
104     a = 255;
105     r = 255;
106     g = 255;
107     b = 255;
108     r *= coeff;
109     g *= coeff;
110     b *= coeff;
111     return r * R_CHANNEL + g * G_CHANNEL + b * B_CHANNEL + a *
112         → A_CHANNEL;
113 }
114
115 int draw_distance_circle(double x, double y) {
116     return implicit_distance(sqrt(x * x + y * y) - 1);
117 }
118
119 int draw_distance_square(double x, double y) {
120     if(fabs(x) > 1 && fabs(y) > 1) {
121         double d = x * x + y * y - 2 * (fabs(x) + fabs(y)) + 2;
122         return implicit_distance(sqrt(d));
123     }
124     return implicit_distance(fmax(fabs(x), fabs(y)) - 1);
125 }
126
127 int draw_circle(double x, double y) {
128     return simple_distance(sqrt(x * x + y * y) - 1);
129 }
130
131 int draw_square(double x, double y) {
132     return simple_distance(fmax(fabs(x), fabs(y)) - 1);
133 }
134
135 int draw_mandelbrot(double x, double y) {
136     double cx = x, cy = y;
137     double nx, ny;
138     int i;
139     for(i = 0; i < max_iter && x * x + y * y < 4; ++i) {
140         nx = x * x - y * y + cx;
141         ny = 2 * x * y + cy;

```

```
141     x = nx;
142     y = ny;
143 }
144 return factual_color((double)i / max_iter);
145 }

146 int draw_julia(double x, double y) {
147     double cx = jx, cy = jy;
148     double nx, ny;
149     int i;
150     for(i = 0; i < max_iter && x * x + y * y < 4; ++i) {
151         nx = x * x - y * y + cx;
152         ny = 2 * x * y + cy;
153         x = nx;
154         y = ny;
155     }
156     return factual_color((double)i / max_iter);
157 }
158 }

159 int draw_burning_ship(double x, double y) {
160     double cx = x, cy = y;
161     double nx, ny;
162     int i;
163     x = y = 0;
164     for(i = 0; i < max_iter && x * x + y * y < 4; ++i) {
165         nx = x * x - y * y + cx;
166         ny = 2 * fabs(x) * fabs(y) + cy;
167         x = nx;
168         y = ny;
169     }
170     return factual_color((double)i / max_iter);
171 }
172 }

173 /* Multiplication de nombres complexes : */
174 /* utilisation : */
175 /* double ax, ay; */
176 /* double bx, by; */
177 /* double rx, ry; */
178 /* complex_mul(ax, ay, bx, by, &rx, &ry); */
179 void complex_mul(double a, double b, double x, double y, double *
180 ← rx, double * ry) {
```

```

181     *rx = a * x - b * y;
182     *ry = a * y + b * x;
183 }
184
185 /* Division de nombres complexes : */
186 /* utilisation : */
187 /* double ax, ay; */
188 /* double bx, by; */
189 /* double rx, ry; */
190 /* complex_div(ax, ay, bx, by, &rx, &ry); */
191 void complex_div(double a, double b, double x, double y, double *
→ rx, double * ry) {
192     double norm = x * x + y * y;
193     complex_mul(a, b, x, -y, rx, ry);
194     *rx /= norm;
195     *ry /= norm;
196 }
197
198 /* Puissance de nombres complexes : */
199 /* utilisation : */
200 /* double ax, ay; */
201 /* int n; */
202 /* double rx, ry; */
203 /* complex_pow(ax, ay, n, &rx, &ry); */
204 void complex_pow(double a, double b, int n, double * rx, double *
→ ry) {
205     *rx = 1;
206     *ry = 0;
207     while(n > 0) {
208         if(n % 2 == 1) {
209             complex_mul(*rx, *ry, a, b, rx, ry);
210         }
211         complex_mul(a, b, a, b, &a, &b);
212         n /= 2;
213     }
214 }
215
216 int color_newton(double x, double y, double coeff) {
217     double sx, sy;
218     double w, sw = 0;
219     double r = 0, g = 0, b = 0;

```

```
220 double cr, cg, cb;
221 int i;
222 for(i = 0; i < newton_n; ++i) {
223     sx = cos(i * 2. * M_PI / newton_n);
224     sy = sin(i * 2. * M_PI / newton_n);
225     w = 1 / (1e-6 + fabs(sx - x) + fabs(sy - y));
226     switch(i % 6) {
227         case 0 : cr = 1; cg = 0; cb = 0; break;
228         case 1 : cr = 0; cg = 1; cb = 0; break;
229         case 2 : cr = 0; cg = 0; cb = 1; break;
230         case 3 : cr = 1; cg = 1; cb = 0; break;
231         case 4 : cr = 0; cg = 1; cb = 1; break;
232         case 5 : cr = 1; cg = 0; cb = 1; break;
233     }
234     r += w * cr;
235     g += w * cg;
236     b += w * cb;
237     sw += w;
238 }
239 unsigned char vr, vg, vb, va;
240 va = 255;
241 vr = coeff * 255 * (r / sw);
242 vg = coeff * 255 * (g / sw);
243 vb = coeff * 255 * (b / sw);
244 return vr * R_CHANNEL + vg * G_CHANNEL + vb * B_CHANNEL + va *
245     → A_CHANNEL;
246 }
247 int draw_newton(double x, double y) {
248     double nx, ny;
249     double znx, zny;
250     double zn1x, zn1y;
251     int i;
252     for(i = 0; i < max_iter && fabs(x * x + y * y - 1) > 1e-6; ++i)
253     → {
254         complex_pow(x, y, newton_n, &znx, &zny);
255         complex_pow(x, y, newton_n - 1, &zn1x, &zn1y);
256         znx -= 1;
257         complex_div(znx, zny, zn1x, zn1y, &nx, &ny);
258         x -= nx / newton_n;
259         y -= ny / newton_n;
```

```

259 }
260     return color_newton(x, y, 1. / log(1 + i));
261 }
262
263 void affichage(SDL_Surface * ecran) {
264     /* Remplissage de l'écran par un gris foncé uniforme : */
265     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
266     → 102));
267     unsigned int color;
268     unsigned char r, g, b, a;
269     SDL_Rect pixel;
270     double x, y;
271     int i, j;
272     for(i = 0; i < largeur; i += resolution) {
273         for(j = 0; j < hauteur; j += resolution) {
274             x = ecran_depuis_camera_x(i + 0.5 * resolution);
275             y = ecran_depuis_camera_y(j + 0.5 * resolution);
276             color = fonction(current_function)(camera_depuis_ecran_x(i),
277             → camera_depuis_ecran_y(j));
278             r = (color / R_CHANNEL) % 256;
279             g = (color / G_CHANNEL) % 256;
280             b = (color / B_CHANNEL) % 256;
281             a = (color / A_CHANNEL) % 256;
282             if(a < 127) continue;
283             pixel.x = i;
284             pixel.y = j;
285             pixel.w = resolution;
286             pixel.h = resolution;
287             SDL_FillRect(ecran, &pixel, SDL_MapRGB(ecran->format, r, g,
288             → b));
289         }
290     }
291
292     stringRGBA(ecran, 10, 10, fonction_name(current_function), 204,
293     → 204, 153, 255);
294     char buffer[1024];
295     sprintf(buffer, "Iterations : %d", max_iter);
296     stringRGBA(ecran, 10, 25, buffer, 204, 204, 153, 255);
297 }
298
299 const char * fonction_name(int id) {
300

```

```
296     switch(id) {
297         case 1 : return "carre";
298         case 2 : return "distance cercle";
299         case 3 : return "distance carre";
300         case 4 : return "Mandelbrot";
301         case 5 : return "Julia";
302         case 6 : return "Burning ship";
303         case 7 : return "Newton";
304         default : return "cercle";
305     }
306 }
307
308 int (* fonction(int id))(double x, double y) {
309     switch(id) {
310         case 1 : return draw_square;
311         case 2 : return draw_distance_circle;
312         case 3 : return draw_distance_square;
313         case 4 : return draw_mandelbrot;
314         case 5 : return draw_julia;
315         case 6 : return draw_burning_ship;
316         case 7 : return draw_newton;
317         default : return draw_circle;
318     }
319 }
320
321 int main() {
322     srand(time(NULL));
323     /* Création d'une fenêtre SDL : */
324     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
325         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
326         exit(EXIT_FAILURE);
327     }
328     SDL_Surface * ecran = NULL;
329     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
330     → | SDL_DOUBLEBUF)) == NULL) {
331         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
332             → SDL_GetError());
333         SDL_Quit();
334         exit(EXIT_FAILURE);
335     }
336     SDL_WM_SetCaption(titre, NULL);
```

```
335      /* Placement du joueur au centre de la fenêtre : */
336
337      cx = 0;
338      cy = 0;
339      cz = 10;
340
341      int active = 1;
342      SDL_Event event;
343      int grab = 0;
344      int refresh = 1;
345
346
347      while(active) {
348
349          if(refresh) {
350              affichage(ecran);
351              SDL_Flip(ecran);
352              refresh = 0;
353          }
354
355          while(SDL_PollEvent(&event)) {
356
357              switch(event.type) {
358                  /* Utilisateur clique sur la croix de la fenêtre : */
359                  case SDL_QUIT : {
360                      active = 0;
361                  } break;
362
363                  /* Utilisateur enfonce une touche du clavier : */
364                  case SDL_KEYDOWN : {
365                      switch(event.key.keysym.sym) {
366                          /* Touche Echap : */
367                          case SDLK_ESCAPE : {
368                              active = 0;
369                          } break;
370
371                          case SDLK_j : {
372                              jx -= 0.025;
373                              refresh = 1;
374                          } break;
375                          case SDLK_l : {
```

```
376         jx += 0.025;
377         refresh = 1;
378     } break;
379     case SDLK_i : {
380         jy += 0.025;
381         refresh = 1;
382     } break;
383     case SDLK_k : {
384         jy -= 0.025;
385         refresh = 1;
386     } break;

387     case SDLK_y : {
388         max_iter += 5;
389         refresh = 1;
390     } break;
391     case SDLK_h : {
392         max_iter -= 5;
393         refresh = 1;
394     } break;

395     case SDLK_t : {
396         newton_n++;
397         refresh = 1;
398     } break;
399     case SDLK_g : {
400         newton_n--;
401         refresh = 1;
402     } break;
403 }
404 }
405 }
406 } break;

407 case SDL_KEYUP : {
408     switch(event.key.keysym.sym) {
409         case SDLK_UP : {
410             current_function = (current_function + 1) %
411                             functions_count;
412             refresh = 1;
413         } break;

414         case SDLK_DOWN : {
```

```
416         current_function = (current_function +
417             → functions_count - 1) % functions_count;
418         refresh = 1;
419     } break;
420 }
421 } break;

422 case SDL_MOUSEBUTTONDOWN : {
423     switch(event.button.button) {
424         case SDL_BUTTON_WHEELUP : {
425             cz *= 0.8;
426             if(cz < 1e-9) {
427                 cz = 1e-9;
428             }
429             refresh = 1;
430         } break;
431
432         case SDL_BUTTON_WHEELDOWN : {
433             cz /= 0.8;
434             if(cz > 1e9) {
435                 cz = 1e9;
436             }
437             refresh = 1;
438         } break;
439
440         case SDL_BUTTON_LEFT : {
441             grab = 1;
442         } break;
443     }
444 } break;

446 case SDL_MOUSEBUTTONUP : {
447     switch(event.button.button) {
448         case SDL_BUTTON_LEFT : {
449             grab = 0;
450         } break;
451     }
452 } break;

454 case SDL_MOUSEMOTION : {
455     if(grab) {
```

```
456         cx += camera_depuis_ecran_z(-event.motion.xrel);
457         cy += camera_depuis_ecran_z(-event.motion.yrel);
458         refresh = 1;
459     }
460     } break;
461 }
462 }
463
464     SDL_Delay(1000 / 60);
465 }
466
467     SDL_FreeSurface(ecran);
468     SDL_Quit();
469     exit(EXIT_SUCCESS);
470 }
```

A.6 Fonctions

A.6.1 Cours

Correction 46 (★ Définir une fonction).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int carre(int x) {
5     return x * x;
6 }
7
8 int main() {
9     int a = 41;
10    printf("(%d + 1) * (%d + 1) = %d\n", a, a,
11           (a + 1) * (a + 1));
12    printf("carre(%d + 1) = %d\n", a, carre(a + 1));
13    if((a + 1) * (a + 1) == carre(a + 1))
14        printf("Bravo !\n");
15    return 0;
16 }
```

A.6.2 Entraînement

Correction 47 (★ Fonction de menu).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int menu() {
5     /* À vous de jouer */
6     int choix;
7     do {
8         printf("Menu : \n 1 - un truc\n 2 - un autre\n 3 -
9             quitter\n---\nVotre choix : ");
10        scanf("%d", &choix);
11    } while (choix < 1 || choix > 3);
12    return choix;
13 }
```

```
12 }
13
14 int main() {
15     printf("%d, bien reçu.\n", menu());
16     exit(EXIT_SUCCESS);
17 }
```

Correction 48 (★★ Calcul moyenne).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* déclaration de la fonction moyenne : */
5 float moyenne();
6
7 int main() {
8     printf("La moyenne de ");
9     /* appel de la fonction moyenne : */
10    printf("= %g\n", moyenne());
11    exit(EXIT_SUCCESS);
12 }

/* Version boucle forever : */
/* définition de la fonction moyenne : */
13 float moyenne() {
14     /* variables locales à la fonction moyenne : */
15     float val;
16     int nb = 0;
17     float sum = 0;
18     /* traite les valeurs lues au clavier */
19     for(;;) {
20         scanf("%f", &val);
21         /* sauf si on trouve une valeur négative */
22         if(val < 0) {
23             break;
24         }
25         /* la valeur est comptabilisée et ajoutée à la somme */
26         sum += val;
27         ++nb;
28     }
29 }
```

```

18 }
19 if(nb > 0) {
20     sum /= nb;
21 }
22 return sum;
23 }
```

```

1 /* Version boucle while : */
2 /* Défaut: nécessite une première entrée valide */
3 /* (ceci duplique le code de saisie). */
4 float moyenne() {
5     float val;
6     int nb = 0;
7     float sum = 0;
8     /* Lecture d'une première entrée : */
9     scanf("%f", &val);
10    /* Boucle tant que : */
11    /* Se lance et continue tant qu'une entrée valide est reçue. */
12    while(val >= 0) {
13        /* Traitement des entrées : */
14        sum += val;
15        ++nb;
16        /* Lecture des entrées suivantes : */
17        scanf("%f", &val);
18    }
19    /* Traitement final des données : */
20    if(nb > 0) {
21        sum /= nb;
22    }
23    return sum;
24 }
```

```

1 /* Version boucle do-while : */
2 /* Défaut : nécessite de dupliquer la condition d'arrêt. */
3 float moyenne() {
4     float val;
5     int nb = 0;
6     float sum = 0;
7     /* Boucle infinie : */
8     do {
```

```
9  /* Lecture des entrées : */
10 scanf("%f", &val);
11 /* Condition d'arrêt dupliquée */
12 if(val >= 0) {
13     /* Traitement des entrées : */
14     sum += val;
15     ++nb;
16 }
17 } while(val >= 0);
18 /* Traitement final des données : */
19 if(nb > 0) {
20     sum /= nb;
21 }
22 return sum;
23 }
```

Correction 49 (★★ Utilisation variables locales).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Une erreur de compilation survenait pour redéfinition de la */
5 /* variable "res". Ceci arrivait dans le scope du switch car */
6 /* par scope chaque variable doit pouvoir s'identifier par son */
7 /* nom. Dans le cas de scopes imbriqués, une variable de même */
8 /* nom peut être déclarée et masque celle du scope parent. */
9
10 int main() {
11     int first, second;
12     char res;
13     printf(">>> ");
14     scanf("%d %c %d", &first, &res, &second);
15     switch(res) {
16         case '+': {
17             /* int res vit dans le scope relatif au case '+' et */
18             /* masque char res */
19             int res = first + second;
20             printf(" = %d\n", res);
21         } break;
22     }
```

```

23     case '*' : {
24         /* long res vit dans le scope relatif au case '*' et      */
25         /* masque char res                                         */
26         long res = (long)first * second;
27         printf(" = %ld\n", res);
28     } break;
29
30     case '/' : {
31         /* double res vit dans le scope relatif au case '/' et      */
32         /* masque char res                                         */
33         double res = (double)first / second;
34         printf(" = %lg\n", res);
35     } break;
36 }
37 exit(EXIT_SUCCESS);
38 }
```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Le code relatif aux cases du switch est copiés dans des      */
5 /* fonctions. Ces opérations sont maintenant locales aux      */
6 /* fonction indépendamment des noms dans la fonction          */
7 /* principale. Ces traitements nécessitent cependant de       */
8 /* l'information de main : valeurs des deux opérandes        */
9 /* (l'opération appartient à la logique du code de main).    */
10 /* int first et int second des fonctions sont les paramètres */
11 /* des fonctions. Ces variables permettent de copier les    */
12 /* valeurs passées en argument à l'appel. Puis, elles peuvent */
13 /* ensuite être utilisées comme variables locales aux        */
14 /* fonctions. */
15
16 void afficher_addition(int first, int second) {
17     int res = first + second;
18     printf(" = %d\n", res);
19 }
20
21 void afficher_multiplication(int first, int second) {
22     long res = (long)first * second;
23     printf(" = %ld\n", res);
```

```
24 }
25
26 void afficher_division(int first, int second) {
27     double res = (double)first / second;
28     printf(" = %lg\n", res);
29 }
30
31 int main() {
32     int first, second;
33     char res;
34     printf(">>> ");
35     scanf("%d %c %d", &first, &res, &second);
36     switch(res) {
37         case '+':
38             afficher_addition(first, second);
39             break;
40
41         case '*':
42             afficher_multiplication(first, second);
43             break;
44
45         case '/':
46             afficher_division(first, second);
47             break;
48     }
49     exit(EXIT_SUCCESS);
50 }
```

Correction 50 (★★★ Compléter fonctions).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Lecture d'un entier, recommence en cas d'erreur. */
5 int scanInt() {
6     int res = 0;
7     /* scanf renvoie le nombre de formats correctement lus. */
8     while(scanf("%d", &res) != 1) {
9         printf("\n[Info] : Un entier sur %lu octets est attendu.\n",
10            → sizeof(int));
```

```

10  /* Pour débloquer la lecture de scanf à la ligne suivante, */
11  /* on peut consommer la ligne actuelle. */
12  while(getchar() != '\n');
13 }
14 return res;
15 }

16
17 /* Liste de caractères valides : permet l'anticipation de      */
18 /* l'erreur en amont.                                         */
19 int carIsOp(char op) {
20     return '+' == op
21     || '-' == op
22     || '/' == op
23     || '*' == op
24     || '%' == op
25     || '^' == op;
26
27 }
28

29 /* Lecture d'un opérateur valide. */
30 char scanOp() {
31     char res = 0;
32     int valid = 0;
33     do {
34         scanf(" %c", &res);
35         valid = carIsOp(res);
36         if(! valid) {
37             printf("\n[Info] : Opérateur incorrect. Attendu +, -, *, /,
38             ↪ %% ou ^.\n");
39         }
40     } while(! valid);
41     return res;
42 }

43 /* Opérations : */
44

45 long addInt(int first, int second) {
46     return (long)first + second;
47 }

48 long subInt(int first, int second) {

```

```

50     return (long)first - second;
51 }
52
53 long mulInt(int first, int second) {
54     return (long)first * second;
55 }
56
57 double divInt(int first, int second) {
58     if(second == 0) {
59         printf("\n[Info] : division par 0.\n");
60         /* Le choix est d'informer l'utilisateur mais non de      */
61         /* corriger l'erreur.                                         */
62     }
63     return (double)first / second;
64 }
65
66 int modInt(int first, int second) {
67     if(second == 0) {
68         printf("\n[Info] : division par 0.\n");
69         /* Le choix est d'informer l'utilisateur mais non de      */
70         /* corriger l'erreur.                                         */
71     }
72     return first % second;
73 }
74
75 long powInt(int first, int second) {
76     if(second < 0) {
77         return 0;
78     }
79     long res = 1;
80     for(; second > 0; --second) {
81         res *= first;
82     }
83     return res;
84 }
85
86 void compute(int first, char op, int second) {
87     switch(op) {
88
89         case '+': {
90             printf(" = %ld\n", addInt(first, second));

```

```

91 } break;
92
93 case '-' : {
94     printf(" = %ld\n", subInt(first, second));
95 } break;
96
97 case '*' : {
98     printf(" = %ld\n", mulInt(first, second));
99 } break;
100
101 case '/' : {
102     printf(" = %lg\n", divInt(first, second));
103 } break;
104
105 case '%' : {
106     printf(" = %d\n", modInt(first, second));
107 } break;
108
109 case '^' : {
110     printf(" = %ld\n", powInt(first, second));
111 } break;
112 }
113 }
114
115 int main() {
116     int first, second;
117     char op;
118     printf(">> ");
119     first = scanInt();
120     op = scanOp();
121     second = scanInt();
122     compute(first, op, second);
123     exit(EXIT_SUCCESS);
124 }
```

Correction 51 (★★★ Calculatrice avec mémoire : séparation en fonctions).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
```

```
4  /* Effectue le calcul entre first et second pour un opérateur */
5  /* donné par le caractère op */
6  /* Renvoie le résultat du calcul (first si l'opérateur n'est pas connu, second si '=') */
7  /* */
8  int applyOp(int first, int second, char op);
9
10 /* Affiche le menu dans la console */
11 void printMenu();
12
13 /* Gère l'interaction du choix de l'option par l'utilisateur */
14 int choixMenu();
15
16 /* lance l'option calculer : renvoie le résultat */
17 int optionCalculer();
18
19 /* lance l'option modifier la variable : renvoie la nouvelle valeur de la variable */
20 /* valeur de la variable */
21 int optionModifier(int variable);
22
23 /* lance l'option d'affichage de la variable */
24 void optionAfficher(int variable);
25
26 /* lance et gère la calculette */
27 void runCalculette();
28
29 int main() {
30     runCalculette();
31     printf("Au revoir.\n");
32     exit(EXIT_SUCCESS);
33 }
34
35 int applyOp(int first, int second, char op) {
36     switch(op) {
37         case '+': return first + second;
38         case '-': return first - second;
39         case '*': return first * second;
40         case '/': return first / second;
41         case '%': return first % second;
42         case '=': return second;
43     }
44     printf("Opérateur \'%c\' non connu.\n", op);
```

```
45     return first;
46 }
47
48 void printMenu() {
49     printf("1 - Calculer\n2 - Modifier variable\n3 - Voir
50         → variable\n0 - Quitter\n---\n");
51 }
52
53 int choixMenu() {
54     int choix;
55     printMenu();
56     printf("Votre choix : ");
57     scanf("%d", &choix);
58     return choix;
59 }
60
61 int optionCalculer() {
62     int resultat;
63     int first, second;
64     char op;
65     printf(">> ");
66     scanf("%d %c %d", &first, &op, &second);
67     resultat = applyOp(first, second, op);
68     printf(">> %d %c %d = %d\n", first, op, second, resultat);
69     return resultat;
70 }
71
72 int optionModifier(int variable) {
73     int second;
74     char op;
75     printf(">> variable = %d ", variable);
76     scanf(" %c %d", &op, &second);
77     variable = applyOp(variable, second, op);
78     printf(">> variable = %d\n", variable);
79     return variable;
80 }
81
82 void optionAfficher(int variable) {
83     printf(">> variable = %d\n", variable);
84 }
```

```
85 void runCalculette() {
86     int variable = 0;
87     int choix;
88     while(choix = choixMenu()) {
89         switch(choix) {
90             case 1 : optionCalculer(); break;
91             case 2 : variable = optionModifier(variable); break;
92             case 3 : optionAfficher(variable); break;
93             default : break;
94         }
95     }
96 }
```

Correction 52 (★★★ Exponentiation et performances).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 long puissance(long a, long n) {
6     long res = 1;
7     long i;
8     for(i = 0; i < n; ++i) {
9         res *= a;
10    }
11    return res;
12 }
13
14 long puissance_rapide(long a, long n) {
15     long res = 1;
16     while(n > 0) {
17         if(n % 2 == 1) {
18             res = res * a;
19         }
20         a *= a;
21         n /= 2;
22     }
23     return res;
24 }
```

```

26 unsigned long puissance_modulo(unsigned long a, unsigned long n,
27   → unsigned long p) {
28   unsigned long res = 1;
29   unsigned long i;
30   for(i = 0; i < n; ++i) {
31     res = (res * a) % p;
32   }
33   return res;
34 }
35 unsigned long puissance_rapide_modulo(unsigned long a, unsigned
36   → long n, unsigned long p) {
37   unsigned long res = 1;
38   while(n > 0) {
39     if(n % 2 == 1) {
40       res = (res * a) % p;
41     }
42     a = (a * a) % p;
43     n /= 2;
44   }
45   return res;
46 }
47 int main() {
48   printf("3 ** 5 = %ld\n", puissance(3, 5));
49   printf("3 ** 5 = %ld\n", puissance_rapide(3, 5));
50   printf("3 ** 5 = %ld\n", puissance_modulo(3, 6, 7));
51   printf("3 ** 5 = %ld\n", puissance_rapide_modulo(3, 6, 7));
52   clock_t start, end;
53   unsigned long a = 42;
54   unsigned long n = 2460320538;
55   unsigned long p = 4285404239;
56   unsigned long res;
57   start = clock();
58   res = puissance_modulo(a, n, p);
59   end = clock();
60   printf("puissance_modulo(%lu, %lu, %lu) = %lu, temps : %g s\n",
61     → a, n, p, res, (double)(end - start) / CLOCKS_PER_SEC);
62   start = clock();
63   res = puissance_rapide_modulo(a, n, p);
64   end = clock();

```

```
64     printf("puissance_rapide_modulo(%lu, %lu, %lu) = %lu, temps : %g\n"
65         "    ↳ s\\n", a, n, p, res, (double)(end - start) / CLOCKS_PER_SEC);
66     exit(EXIT_SUCCESS);
}
```

Correction 53 (★★★ Affrontement de personnages).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 /* Statistiques générales en début d'une partie : */
6
7 int max_vie = 100;
8 int start_atk = 50;
9 int start_def = 20;
10
11 /* Statistiques du joueur : */
12
13 int self_vie;
14 int self_atk;
15 int self_def;
16
17 /* Statistiques de l'adversaire : */
18
19 int adv_vie;
20 int adv_atk;
21 int adv_def;
22
23 /* Ici l'utilisation de variables globales se justifie par le */
24 /* besoin de séparer en fonctions, mais le manque de           */
25 /* connaissances à l'étape actuelle du cours pour modifier   */
26 /* plusieurs variables avec des pointeurs / renvoyer plusieurs */
27 /* valeurs.                                                 */
28
29 /* Mise aux valeurs initiales des statistiques des deux      */
30 /* joueurs.                                                 */
31 void init() {
32     self_vie = max_vie;
33     self_atk = start_atk;
```

```
34     self_def = start_def;
35
36     adv_vie = max_vie;
37     adv_atk = start_atk;
38     adv_def = start_def;
39 }
40
41 /* Fonction générale pour calculer les dégats d'une attaque.*/
42 int degats(int tori_atk, int uke_def) {
43     int res = tori_atk / uke_def;
44     if(res < 1) {
45         res = 1;
46     }
47     return res;
48 }
49
50 /* Fonction générale pour appliquer un buff : en pourcentage */
51 int boost(int value, int pourcentage) {
52     value = ((long)(100 + pourcentage) * value) / 100;
53     if(value < 1) {
54         value = 1;
55     }
56     return value;
57 }
58
59 /* Fonctions relatives au joueur : */
60
61 void self_regen() {
62     self_vie += 0.25 * max_vie;
63     if(self_vie > max_vie) {
64         self_vie = max_vie;
65     }
66 }
67
68 void self_hit() {
69     adv_vie -= degats(self_atk, adv_def);
70 }
71
72 void self_boost_atk() {
73     self_atk = boost(self_atk, 20);
74 }
```

```
75
76 void self_boost_def() {
77     self_def = boost(self_def, 30);
78 }
79
80 void self_action(int id) {
81     switch(id) {
82         case 1 : {
83             printf("Le joueur cogne l'\ adversaire.\n");
84             self_hit();
85         } break;
86
87         case 2 : {
88             printf("Le joueur se soigne.\n");
89             self_regen();
90         } break;
91
92         case 3 : {
93             printf("Le joueur augmente son attaque.\n");
94             self_boost_atk();
95         } break;
96
97         case 4 : {
98             printf("Le joueur augmente sa defense.\n");
99             self_boost_def();
100        } break;
101    }
102 }
103
104 /* Par manque de générnicité et l'utilisation de variables */  
/* globales, le code est dupliqué : */  

105
106 /* Fonctions relatives à l'adversaire : */
107
108 void adv_regen() {
109     adv_vie += 0.25 * max_vie;
110     if(adv_vie > max_vie) {
111         adv_vie = max_vie;
112     }
113 }
114
115
```

```
116 void adv_hit() {
117     self_vie -= degats(adv_atk, self_def);
118 }
119
120 void adv_boost_atk() {
121     adv_atk = boost(adv_atk, 20);
122 }
123
124 void adv_boost_def() {
125     adv_def = boost(adv_def, 30);
126 }
127
128 void adv_action(int id) {
129     switch(id) {
130         case 1 : {
131             printf("L\\'adversaire cogne le joueur.\n");
132             adv_hit();
133         } break;
134
135         case 2 : {
136             printf("L\\'adversaire se soigne.\n");
137             adv_regen();
138         } break;
139
140         case 3 : {
141             printf("L\\'adversaire augmente son attaque.\n");
142             adv_boost_atk();
143         } break;
144
145         case 4 : {
146             printf("L\\'adversaire augmente sa defense.\n");
147             adv_boost_def();
148         } break;
149     }
150 }
151
152 /* Affichage du jeu : */
153
154 void display_actions() {
155     printf(" 1 - cogner.\n"
156           " 2 - se soigner.\n"
```

```
157         " 3 - augmenter attaque.\n"
158         " 4 - augmenter defense.\n");
159     }
160
161     void display() {
162         printf("-----+-----+\n");
163         printf("|      Joueur      |      Adversaire      |\n");
164         printf("-----+-----+\n");
165         printf(" | Vie : %11d | Vie : %11d |\n", self_vie, adv_vie);
166         printf(" | Attaque : %7d | Attaque : %7d |\n", self_atk,
167             → adv_atk);
168         printf(" | Defense : %7d | Defense : %7d |\n", self_def,
169             → adv_def);
170         printf("-----+-----+\n");
171     }
172
173     /* Un combat : */
174
175     int main() {
176         int self_choix;
177         int adv_choix;
178         srand(time(NULL));
179         init();
180         while(self_vie > 0 && adv_vie > 0) {
181             display();
182             display_actions();
183             printf("Votre choix : ");
184             scanf("%d", &self_choix);
185             adv_choix = 1;
186             if(rand() % 2 == 0) {
187                 adv_choix = 2 + rand() % 3;
188             }
189             self_action(self_choix);
190             adv_action(adv_choix);
191         }
192         if(self_vie <= 0 && adv_vie <= 0) {
193             printf("Match nul.\n");
194         } else if(self_vie > 0) {
195             printf("Victoire du joueur.\n");
196         } else {
197             printf("Defaite du joueur.\n");
198         }
199     }
200 }
```

```

196 }
197 exit(EXIT_SUCCESS);
198 }
```

Correction 54 (★★★ Racine carré par dichotomie).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef COMPARE
4 #include <math.h>
5 #endif
6 #include <time.h>
7
8 double equation(double x, double target) {
9     return x * x - target;
10 }
11
12 double dicho_sqrt(double x) {
13     if(((x > 0) ? x : -x) < 1e-30) {
14         return 0.;
15     }
16     double start = 0;
17     double end = (x > 1) ? x : 1;
18     double mid;
19     int i;
20     for(i = 0; i < 256; ++i) {
21         mid = 0.5 * (start + end);
22         if(equation(start, x) * equation(mid, x) < 0) {
23             end = mid;
24         } else if(equation(end, x) * equation(mid, x) < 0) {
25             start = mid;
26         } else {
27             return mid;
28         }
29     }
30     return mid;
31 }
32
33 int main() {
34     clock_t start, end;
```

```
35 double x;
36 double target;
37 printf("Entrez un réel : ");
38 scanf("%lf", &target);
39 start = clock();
40 x = dicho_sqrt(target);
41 end = clock();
42 printf("La racine carré de %g est %.16f (en %.6f s)\n", target,
43     ↪ x, (double)(end - start) / CLOCKS_PER_SEC);
44 #ifdef COMPARE
45     start = clock();
46     x = sqrt(target);
47     end = clock();
48     printf("La racine carré de %g est %.16f (en %.6f s) (d'après
49         ↪ math)\n", target, x, (double)(end - start) /
50             ↪ CLOCKS_PER_SEC);
51 #endif
52     exit(EXIT_SUCCESS);
53 }
```

Correction 55 (★★★ Calcul coefficients binomiaux).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int binom(int n, int k) {
5     if(k < 0 || k > n) return 0;
6     if(k == 0 || k == n) return 1;
7     return binom(n - 1, k - 1) + binom(n - 1, k);
8 }
9
10 int binom_fast(int n, int k) {
11     if(k < 0 || k > n) return 0;
12     if(k == 0 || k == n) return 1;
13     int res = n;
14     int i;
15     for(i = 1; i < k; ++i) {
16         res *= n - i;
17         res /= (i + 1);
18     }
}
```

```

19     return res;
20 }
21
22 int main() {
23     int n;
24     printf("Entrez l'entier n : ");
25     scanf("%d", &n);
26     int i, k;
27     for(i = 0; i <= n; ++i) {
28         printf("%2d : ", i);
29         for(k = 0; k <= i; ++k) {
30             printf(" %3d", binom_fast(i, k));
31         }
32         printf("\n");
33     }
34     exit(EXIT_SUCCESS);
35 }
```

Correction 56 (★★★ Développement de Taylor : calcul de sinus).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #ifdef COMPARE
4 #include <math.h>
5 #endif
6 #include <time.h>
7
8 #define PI 3.14159265359
9 #define PIL 3.141592653589793l
10
11 double newton_sqrt(double target) {
12     double x;
13     x = target / 2.;
14     int i;
15     for(i = 0; i < 32; ++i) {
16         x = (x * x + target) / (2 * x);
17     }
18     return x;
19 }
```

```
21 double taylor_sin_fast(double x) {
22     int sign = 1;
23     x -= PI * 2 * (int)(x / (PI * 2));
24     if(x < 0) x += PI * 2;
25     if(x > PI) x -= PI * 2;
26     if(x < 0) {
27         sign = -1;
28         x *= -1;
29     }
30     if(x > PI / 2.) x = PI / 2. - (x - PI / 2.);
31     double y = x * (1 + x * x * (-1. / 6. + x * x * (1. / 120.
32     ↪ - x * x / 5040.)));
33     return sign * y;
34 }
35
36 long double taylor_sin_precise(long double x) {
37     int sign = 1;
38     x -= PIL * 2 * (long)(x / (PIL * 2));
39     if(x < 0) x += PIL * 2;
40     if(x > PIL) x -= PIL * 2;
41     if(x < 0) {
42         sign = -1;
43         x *= -1;
44     }
45     if(x > PIL / 2.) x = PIL / 2. - (x - PIL / 2.);
46     long double y = x * (1 + x * x * (-1. / 6.1 + x * x * (1. /
47     ↪ / 120.1 + x * x * (-1. / 5040.1 + x * x * (1. /
48     ↪ 362880.1 + x * x * (-1. / 39916800.1 + x * x * (1. /
49     ↪ 6227020800.1 - x * x / 1307674368000.1)))););
50     return sign * y;
51 }
52
53 double taylor_sin_half(double x) {
54     int sign = 1;
55     x -= PI * 2 * (int)(x / (PI * 2));
56     if(x < 0) x += PI * 2;
57     if(x > PI) x -= PI * 2;
58     if(x < 0) {
59         sign = -1;
60         x *= -1;
61     }
62 }
```

```

58     if(x > PI / 2.) x = PI / 2. - (x - PI / 2.);
59     x -= PI / 4.;
60     double y = 0.5 * newton_sqrt(2) * (1. + x * (1. + x * (-1.
61     ↪ / 2. + x * (-1. / 6. + x * (1. / 24. + x * (1. / 120.
62     ↪ + x * (-1. / 720. + x * (-1. / 5040. + x /
63     ↪ 40320.))))));
64     return sign * y;
65 }
66
67 int main() {
68     clock_t start, end;
69     double x, y;
70     long double ly;
71     printf("Entrez un réel : ");
72     scanf("%lf", &x);
73     /*
74     start = clock();
75     y = taylor_sin_half(x);
76     end = clock();
77     printf("sin(%g) = %.16f (en %.6f s) (en 0.5)\n", x, y,
78     ↪ (double)(end - start) / CLOCKS_PER_SEC);
79     */
80     start = clock();
81     y = taylor_sin_fast(x);
82     end = clock();
83     printf("sin(%g) = %.16f (en %.6f s) (rapide en 0)\n", x,
84     ↪ y, (double)(end - start) / CLOCKS_PER_SEC);
85 #ifdef COMPARE
86     start = clock();
87     y = sin(x);
88     end = clock();
89     printf("sin(%g) = %.16f (en %.6f s) (d'après math)\n", x,
90     ↪ y, (double)(end - start) / CLOCKS_PER_SEC);
91 #endif
92     start = clock();
93     ly = taylor_sin_precise(x);
94     end = clock();
95     printf("sinl(%g) = %.16Lf (en %.6f s) (développé en 0)\n",
96     ↪ x, ly, (double)(end - start) / CLOCKS_PER_SEC);
97 #ifdef COMPARE
98     start = clock();
99 
```

```
92     ly = sinl(x);
93     end = clock();
94     printf("sinl(%g) = %.16Lf (en %.6f s) (d'après math)\n",
95            x, ly, (double)(end - start) / CLOCKS_PER_SEC);
96 #endif
97     exit(EXIT_SUCCESS);
}
```

Correction 57 (∞ Contrôle de l'aléatoire).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5 #include <SDL/SDL.h>
6 #include <SDL/SDL_gfxPrimitives.h>
7
8 #define MAX_ENTRIES 42
9
10 /* Paramètres de la fenêtre : */
11 const int largeur = 800;
12 const int hauteur = 600;
13 const char * titre = "ESGI random";
14 SDL_Surface * ecran = NULL;
15
16 const int trials = 10000000;
17 int essai = 0;
18 unsigned long counts[MAX_ENTRIES];
19
20 void draw_random() {
21
22     char buffer[1024];
23     int i;
24     unsigned long max_count = 0;
25     unsigned long sum_count = 0;
26     for(i = 0; i < MAX_ENTRIES; ++i) {
27         max_count = (max_count > counts[i]) ? max_count : counts[i];
28         sum_count += counts[i];
29     }
30     for(i = 0; i < MAX_ENTRIES; ++i) {
```

```

31     boxRGBA(ecran,
32         300, 100 + (i * (hauteur - 200)) / (MAX_ENTRIES + 1),
33         300 + ((largeur - 400) * (counts[i])) / max_count, 100 + ((i
34             + 1) * (hauteur - 200)) / (MAX_ENTRIES + 1),
35         127, 127, 127, 255);
36     rectangleRGBA(ecran,
37         300, 100 + (i * (hauteur - 200)) / (MAX_ENTRIES + 1),
38         300 + ((largeur - 400) * (counts[i])) / max_count, 100 + ((i
39             + 1) * (hauteur - 200)) / (MAX_ENTRIES + 1),
40         255, 255, 255, 255);
41     sprintf(buffer, "P(X = %d) = %g", i, (double)counts[i] /
42         sum_count);
43     stringRGBA(ecran, 10, 100 + ((i + 0.5) * (hauteur - 200)) /
44         (MAX_ENTRIES + 1), buffer, 255, 255, 255, 255);
45 }
46 sprintf(buffer, "%d / %d essais", essai, trials);
47 stringRGBA(ecran, 10, 10, buffer, 255, 255, 255, 255);
48 }

49
50 int uniform_random(int entries) {
51     return rand() % entries;
52 }
53
54 int baised_fair_take_random(int entries) {
55     int i;
56     for(i = 0; i < entries; ++i) {
57         if(rand() % entries == 0) return i;
58     }
59     return entries - 1;
60 }
61
62 int triangular_random(int entries) {
63     int first = uniform_random(entries);
64     int second = uniform_random(entries);
65     return first < second ? first : second;
66 }
67
68 int exponential_random(int entries, const int roundness) {
69     int i;
70     for(;;) {
71         for(i = 0; i < entries; ++i) {

```

```
68     if(rand() % roundness == 0) return i;
69 }
70 }
71 }
72
73 int exponential_high_random(int entries) {
74     return exponential_random(entries, 2);
75 }
76
77 int exponential_medium_random(int entries) {
78     return exponential_random(entries, 3);
79 }
80
81 int exponential_low_random(int entries) {
82     return exponential_random(entries, 10);
83 }
84
85 int normal_random(int entries) {
86     const int splits = 4;
87     int i;
88     long sum = 0;
89     for(i = 0; i < splits; ++i) {
90         sum += uniform_random(entries);
91     }
92     return sum / splits;
93 }
94
95 int gamma_random(int entries) {
96     const int splits = 2;
97     int i;
98     long sum = 0;
99     for(i = 0; i < splits; ++i) {
100        sum += triangular_random(entries);
101    }
102    return sum / splits;
103 }
104
105 int mix_uniform_normal_random(int entries, const double factor) {
106     if(uniform_random(1000) / 1000. < factor) {
107         return uniform_random(entries);
108     }
```

```

109     return normal_random(entries);
110 }
111
112 int mix_un_high_random(int entries) {
113     return mix_uniform_normal_random(entries, 0.8);
114 }
115
116 int mix_un_medium_random(int entries) {
117     return mix_uniform_normal_random(entries, 0.5);
118 }
119
120 int mix_un_low_random(int entries) {
121     return mix_uniform_normal_random(entries, 0.2);
122 }
123
124 int troll_random(int entries) {
125     const int splits = 3;
126     int i;
127     int val = 1;
128     for(i = 0; i < splits; ++i) {
129         val = (uniform_random(entries) * val) % entries;
130     }
131     if(val == 0) return troll_random(entries);
132     return val;
133 }
134
135 void test_random(int entries, int (*random)(int)) {
136     char buffer[1024];
137     int i;
138     for(i = 0; i < MAX_ENTRIES; ++i) {
139         counts[i] = 0;
140     }
141     unsigned long t = 1;
142     int updates = 0;
143     for(essai = 0; essai < trials; ++essai) {
144         ++(counts[random(entries)]);
145         if(essai > t || essai % 10000 == 0) {
146             SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
147             → 51));
148             draw_random();
149             SDL_Flip(ecran);

```

```
149     if(essai > t) {
150         t *= 2;
151         SDL_Delay(1 + 1000 / (++updates));
152     }
153 }
154 }
155 }
156
157 int read_command() {
158     printf("Aléatoires :\n");
159     printf(" 1. uniform random\n");
160     printf(" 2. baised fair take random\n");
161     printf(" 3. triangular random\n");
162     printf(" 4. exponential high random\n");
163     printf(" 5. exponential medium random\n");
164     printf(" 6. exponential low random\n");
165     printf(" 7. normal random\n");
166     printf(" 8. gamma random\n");
167     printf(" 9. mixed high random\n");
168     printf(" 10. mixed medium random\n");
169     printf(" 11. mixed low random\n");
170     printf(" 12. troll random\n");
171
172     printf(" 0. quitter\n");
173     printf(">>> ");
174     int choix;
175     scanf("%d", &choix);
176     switch(choix) {
177         case 0 : return 0;
178         case 1 : test_random(MAX_ENTRIES, uniform_random); break;
179         case 2 : test_random(MAX_ENTRIES, baised_fair_take_random);
180             → break;
181         case 3 : test_random(MAX_ENTRIES, triangular_random); break;
182         case 4 : test_random(MAX_ENTRIES, exponential_high_random);
183             → break;
184         case 5 : test_random(MAX_ENTRIES, exponential_medium_random);
185             → break;
186         case 6 : test_random(MAX_ENTRIES, exponential_low_random);
187             → break;
188         case 7 : test_random(MAX_ENTRIES, normal_random); break;
189         case 8 : test_random(MAX_ENTRIES, gamma_random); break;
```

```

186     case 9 : test_random(MAX_ENTRIES, mix_un_high_random); break;
187     case 10 : test_random(MAX_ENTRIES, mix_un_medium_random);
188     ↪ break;
189     case 11 : test_random(MAX_ENTRIES, mix_un_low_random); break;
190     case 12 : test_random(MAX_ENTRIES, troll_random); break;
191
192     default : break;
193 }
194
195 return 1;
196 }

197 int main() {
198     srand(time(NULL));
199     /* Création d'une fenêtre SDL : */
200     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
201         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
202         exit(EXIT_FAILURE);
203     }
204     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
205     ↪ | SDL_DOUBLEBUF)) == NULL) {
206         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
207             ↪ SDL_GetError());
208         SDL_Quit();
209         exit(EXIT_FAILURE);
210     }
211     SDL_WM_SetCaption(titre, NULL);
212     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
213     ↪ 51));
214     SDL_Flip(ecran);

215     while(read_command()) {
216
217         SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
218         ↪ 51));
219         draw_random();
220         SDL_Flip(ecran);
221         SDL_Delay(1000 / 60);
222     }

223     SDL_FreeSurface(ecran);
224     SDL_Quit();
225 }
```

```
222     exit(EXIT_SUCCESS);  
223 }
```

Correction 58 (∞ Courbes récursives).

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <math.h>  
4 #include <time.h>  
5 #include <SDL/SDL.h>  
6 #include <SDL/SDL_gfxPrimitives.h>  
7  
8 /* Paramètres de la fenêtre : */  
9 const int largeur = 800;  
10 const int hauteur = 800;  
11 const char * titre = "ESGI graph";  
12 const int functions_count = 9;  
13  
14 /* Position de la caméra et échelle de la vue : */  
15 float cx, cy, cz;  
16  
17 int current_function = 0;  
18 int iterations = 2;  
19 int segments = 1;  
20  
21 void (*fonction(int id))(SDL_Surface * ecran, double sx, double  
22    ↴ sy, double ex, double ey, int n, int r, int g, int b, int a);  
23  
24 const char * fonction_name(int id);  
25  
26 int ecran_depuis_camera_x(float x) {  
27     return ((x - cx) * largeur / 2) / cz + largeur / 2;  
28 }  
29  
30 int ecran_depuis_camera_y(float y) {  
31     return ((y - cy) * hauteur / 2) / cz + hauteur / 2;  
32 }  
33  
34 float ecran_depuis_camera_z(int z) {  
35     return (z * largeur / 2.) / cz;
```

```

35 }
36
37 float camera_depuis_ecran_x(int x) {
38     return 2. * ((x - largeur / 2) * cz) / largeur + cx;
39 }
40
41 float camera_depuis_ecran_y(int y) {
42     return 2. * ((y - hauteur / 2) * cz) / hauteur + cy;
43 }
44
45 float camera_depuis_ecran_z(int z) {
46     return (2 * z * cz) / largeur;
47 }
48
49 void affichage(SDL_Surface * ecran) {
50     /* Remplissage de l'écran par un gris foncé uniforme : */
51     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
52         102));
53
54     void (*line)(SDL_Surface * ecran, double sx, double sy, double
55         ex, double ey, int n, int r, int g, int b, int a) =
56         fonction(current_function);
57
58     double scale = 10;
59     if(segments <= 1) {
60         line(ecran,
61             ecran_depuis_camera_x(-scale), ecran_depuis_camera_y(0),
62             ecran_depuis_camera_x(scale), ecran_depuis_camera_y(0),
63             iterations,
64             255, 255, 255, 255);
65     } else {
66         int i;
67         for(i = 0; i < segments; ++i) {
68             line(ecran,
69                 ecran_depuis_camera_x(scale * cos(i * 2 * M_PI /
70                     segments)), ecran_depuis_camera_y(scale * sin(i * 2 *
71                     M_PI / segments)),
72                 ecran_depuis_camera_x(scale * cos((i + 1) * 2 * M_PI /
73                     segments)), ecran_depuis_camera_y(scale * sin((i + 1) *
74                     * 2 * M_PI / segments)),
75                 iterations,
76                 iterations,
77                 iterations,
78                 iterations);
79     }
80 }

```

```
69         255, 255, 255, 255);
70     }
71 }
72
73     char buffer[1024];
74     stringRGB(A(ecran, 10, 10, fonction_name(current_function), 204,
75     ↵ 204, 153, 255);
76     sprintf(buffer, "Iterations : %d", iterations);
77     stringRGB(A(ecran, 10, 25, buffer, 204, 204, 153, 255);
78     sprintf(buffer, "Segments : %d", segments);
79     stringRGB(A(ecran, 10, 40, buffer, 204, 204, 153, 255);
80 }
81
82 void flocon_koch(SDL_Surface * ecran, double sx, double sy, double
83     ↵ ex, double ey, int n, int r, int g, int b, int a) {
84     if(n <= 1) {
85         lineRGB(A(ecran, sx, sy, ex, ey, r, g, b, a);
86         return;
87     }
88     int nx = (ey - sy);
89     int ny = -(ex - sx);
90     flocon_koch(ecran, sx, sy, sx + (ex - sx) / 3, sy + (ey - sy)
91     ↵ / 3, n - 1, r, g, b, a);
92     flocon_koch(ecran, sx + (ex - sx) / 3, sy + (ey - sy) / 3, 0.5
93     ↵ * (sx + ex) + 0.25 * nx, 0.5 * (sy + ey) + 0.25 * ny, n -
94     ↵ 1, r, g, b, a);
95     flocon_koch(ecran, 0.5 * (sx + ex) + 0.25 * nx, 0.5 * (sy +
96     ↵ ey) + 0.25 * ny, sx + 2 * (ex - sx) / 3, sy + 2 * (ey -
97     ↵ sy) / 3, n - 1, r, g, b, a);
98     flocon_koch(ecran, sx + 2 * (ex - sx) / 3, sy + 2 * (ey - sy)
99     ↵ / 3, ex, ey, n - 1, r, g, b, a);
100 }
```

```

101 courbe_koch(ecran, sx, sy, sx + (ex - sx) / 3, sy + (ey - sy)
102   → / 3, n - 1, r, g, b, a);
103 courbe_koch(ecran, sx + (ex - sx) / 3, sy + (ey - sy) / 3, sx
104   → + (ex - sx) / 3 + nx / 3, sy + (ey - sy) / 3 + ny / 3, n -
105   → 1, r, g, b, a);
106 courbe_koch(ecran, sx + (ex - sx) / 3 + nx / 3, sy + (ey - sy)
107   → / 3 + ny / 3, sx + 2 * (ex - sx) / 3 + nx / 3, sy + 2 *
108   → (ey - sy) / 3 + ny / 3, n - 1, r, g, b, a);
109 courbe_koch(ecran, sx + 2 * (ex - sx) / 3 + nx / 3, sy + 2 *
110   → (ey - sy) / 3 + ny / 3, sx + 2 * (ex - sx) / 3, sy + 2 *
111   → (ey - sy) / 3, n - 1, r, g, b, a);
112 courbe_koch(ecran, sx + 2 * (ex - sx) / 3, sy + 2 * (ey - sy)
113   → / 3, ex, ey, n - 1, r, g, b, a);
114 }

115 void courbe_koch_alt(SDL_Surface * ecran, double sx, double sy,
116   → double ex, double ey, int n, int r, int g, int b, int a) {
117   if(n <= 1) {
118     linerRGBAlpha(ecran, sx, sy, ex, ey, r, g, b, a);
119     return;
120   }
121   int nx = (ey - sy);
122   int ny = -(ex - sx);
123   courbe_koch_alt(ecran, sx, sy, sx + (ex - sx) / 4, sy + (ey -
124     → sy) / 4, n - 1, r, g, b, a);
125   courbe_koch_alt(ecran, sx + (ex - sx) / 4, sy + (ey - sy) / 4,
126     → sx + (ex - sx) / 4 + nx / 4, sy + (ey - sy) / 4 + ny / 4,
127     → n - 1, r, g, b, a);
128   courbe_koch_alt(ecran, sx + (ex - sx) / 4 + nx / 4, sy + (ey -
129     → sy) / 4 + ny / 4, sx + 2 * (ex - sx) / 4 + nx / 4, sy + 2
130     → * (ey - sy) / 4 + ny / 4, n - 1, r, g, b, a);
131   courbe_koch_alt(ecran, sx + 2 * (ex - sx) / 4 + nx / 4, sy + 2
132     → * (ey - sy) / 4 + ny / 4, sx + 2 * (ex - sx) / 4 - nx / 4,
133     → sy + 2 * (ey - sy) / 4 - ny / 4, n - 1, r, g, b, a);
134   courbe_koch_alt(ecran, sx + 2 * (ex - sx) / 4 - nx / 4, sy + 2
135     → * (ey - sy) / 4 - ny / 4, sx + 3 * (ex - sx) / 4 - nx / 4,
136     → sy + 3 * (ey - sy) / 4 - ny / 4, n - 1, r, g, b, a);
137   courbe_koch_alt(ecran, sx + 3 * (ex - sx) / 4 - nx / 4, sy + 3
138     → * (ey - sy) / 4 - ny / 4, sx + 3 * (ex - sx) / 4, sy + 3 *
139     → (ey - sy) / 4, n - 1, r, g, b, a);
140   courbe_koch_alt(ecran, sx + 3 * (ex - sx) / 4, sy + 3 * (ey -
141     → sy) / 4, ex, ey, n - 1, r, g, b, a);

```

```
122 }
123
124 void courbe_cesaro(SDL_Surface * ecran, double sx, double sy,
125 → double ex, double ey, int n, int r, int g, int b, int a) {
126 if(n <= 1) {
127     lineRGBA(ecran, sx, sy, ex, ey, r, g, b, a);
128     return;
129 }
130 int nx = (ey - sy);
131 int ny = -(ex - sx);
132 double biais = 0.05;
133 double b1 = 1. + biais;
134 double b2 = 2. - biais;
135 courbe_cesaro(ecran, sx, sy, sx + (ex - sx) / 3, sy + (ey -
136 → sy) / 3, n - 1, r, g, b, a);
courbe_cesaro(ecran, sx + (ex - sx) / 3, sy + (ey - sy) / 3,
137 → sx + b1 * (ex - sx) / 3 + nx / 3, sy + b1 * (ey - sy) / 3
→ + ny / 3, n - 1, r, g, b, a);
courbe_cesaro(ecran, sx + b1 * (ex - sx) / 3 + nx / 3, sy + b1
138 → * (ey - sy) / 3 + ny / 3, sx + b2 * (ex - sx) / 3 + nx /
→ 3, sy + b2 * (ey - sy) / 3 + ny / 3, n - 1, r, g, b, a);
courbe_cesaro(ecran, sx + b2 * (ex - sx) / 3 + nx / 3, sy + b2
139 → * (ey - sy) / 3 + ny / 3, sx + 2 * (ex - sx) / 3, sy + 2 *
→ (ey - sy) / 3, n - 1, r, g, b, a);
courbe_cesaro(ecran, sx + 2 * (ex - sx) / 3, sy + 2 * (ey -
140 → sy) / 3, ex, ey, n - 1, r, g, b, a);
}
141
142 void courbe_dragon(SDL_Surface * ecran, double sx, double sy,
143 → double ex, double ey, int n, int r, int g, int b, int a) {
144 if(n <= 1) {
145     lineRGBA(ecran, sx, sy, ex, ey, r, g, b, a);
146     return;
147 }
148 int nx = (ey - sy);
149 int ny = -(ex - sx);
courbe_dragon(ecran, sx, sy, 0.5 * (sx + ex) - 0.5 * nx, 0.5 *
150 → (sy + ey) - 0.5 * ny, n - 1, r, g, b, a);
courbe_dragon(ecran, ex, ey, 0.5 * (sx + ex) - 0.5 * nx, 0.5 *
→ (sy + ey) - 0.5 * ny, n - 1, r, g, b, a);
}
```

```

151
152     double tx;
153     double ty;
154     double tangle;
155
156     void Tortue_avancer(double value, SDL_Surface * ecran, int r, int
157     → g, int b, int a) {
158         double x = tx, y = ty;
159         tx += value * cos((tangle * M_PI) / 180.);
160         ty += value * sin((tangle * M_PI) / 180.);
161         lineRGBA(ecran, x, y, tx, ty, r, g, b, a);
162     }
163
164     void Tortue_tourner(int angle) {
165         angle %= 360;
166         tangle += angle;
167         if(tangle < 0) tangle += 360;
168         if(tangle > 360) tangle -= 360;
169     }
170
171     void courbe_gosper_rule_A(SDL_Surface * ecran, double value, int
172     → n, int r, int g, int b, int a);
173     void courbe_gosper_rule_B(SDL_Surface * ecran, double value, int
174     → n, int r, int g, int b, int a);
175
176     void courbe_gosper_rule_A(SDL_Surface * ecran, double value, int
177     → n, int r, int g, int b, int a) {
178         if(n <= 1) {
179             Tortue_avancer(value, ecran, r, g, b, a);
180             return;
181         }
182         value /= sqrt(7.);
183         courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
184         Tortue_tourner(-60);
185         courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
186         Tortue_tourner(-60);
187         Tortue_tourner(-60);
188         courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
189         Tortue_tourner(60);
190         courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
191         Tortue_tourner(60);

```

```
188     Tortue_tourner(60);
189     courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
190     courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
191     Tortue_tourner(60);
192     courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
193     Tortue_tourner(-60);
194 }
195
196 void courbe_gosper_rule_B(SDL_Surface * ecran, double value, int
197 → n, int r, int g, int b, int a) {
198     if(n <= 1) {
199         Tortue_avancer(value, ecran, r, g, b, a);
200         return;
201     }
202     value /= sqrt(7.);
203     Tortue_tourner(60);
204     courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
205     Tortue_tourner(-60);
206     courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
207     courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
208     Tortue_tourner(-60);
209     Tortue_tourner(-60);
210     courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
211     Tortue_tourner(-60);
212     courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
213     Tortue_tourner(60);
214     Tortue_tourner(60);
215     courbe_gosper_rule_A(ecran, value, n - 1, r, g, b, a);
216     Tortue_tourner(60);
217     courbe_gosper_rule_B(ecran, value, n - 1, r, g, b, a);
218 }
219
220 void courbe_gosper(SDL_Surface * ecran, double sx, double sy,
221 → double ex, double ey, int n, int r, int g, int b, int a) {
222     double norm = sqrt(pow(ex - sx, 2.) + pow(ey - sy, 2.));
223     double angle = ((fabs(ey - sy) < 1e-9) ? 1 : ((ey - sy) /
224 → fabs(ey - sy))) * acos((ex - sx) / norm) * 180. / M_PI;
225     tx = sx;
226     ty = sy;
227     tangle = angle + n * 19.11;
228     courbe_gosper_rule_A(ecran, norm, n, r, g, b, a);
```

```
226 }
227
228 void courbe_hilbert_rule_L(SDL_Surface * ecran, double value, int
229   → n, int r, int g, int b, int a);
230 void courbe_hilbert_rule_R(SDL_Surface * ecran, double value, int
231   → n, int r, int g, int b, int a);
232
233 void courbe_hilbert_rule_L(SDL_Surface * ecran, double value, int
234   → n, int r, int g, int b, int a) {
235
236   if(n <= 1) {
237     return;
238   }
239   Tortue_tourner(-90);
240   courbe_hilbert_rule_R(ecran, value, n - 1, r, g, b, a);
241   Tortue_avancer(value, ecran, r, g, b, a);
242   Tortue_tourner(+90);
243   courbe_hilbert_rule_L(ecran, value, n - 1, r, g, b, a);
244   Tortue_avancer(value, ecran, r, g, b, a);
245   courbe_hilbert_rule_L(ecran, value, n - 1, r, g, b, a);
246   Tortue_tourner(+90);
247   Tortue_avancer(value, ecran, r, g, b, a);
248   courbe_hilbert_rule_R(ecran, value, n - 1, r, g, b, a);
249   Tortue_tourner(-90);
250 }
251
252 void courbe_hilbert_rule_R(SDL_Surface * ecran, double value, int
253   → n, int r, int g, int b, int a) {
254
255   if(n <= 1) {
256     return;
257   }
258   Tortue_tourner(+90);
259   courbe_hilbert_rule_L(ecran, value, n - 1, r, g, b, a);
260   Tortue_avancer(value, ecran, r, g, b, a);
261   Tortue_tourner(-90);
262   courbe_hilbert_rule_R(ecran, value, n - 1, r, g, b, a);
263   Tortue_avancer(value, ecran, r, g, b, a);
```

```
263     courbe_hilbert_rule_L(ecran, value, n - 1, r, g, b, a);
264     Tortue_tourner(+90);
265 }
266
267 void courbe_hilbert(SDL_Surface * ecran, double sx, double sy,
268 → double ex, double ey, int n, int r, int g, int b, int a) {
269     double norm = sqrt(pow(ex - sx, 2.) + pow(ey - sy, 2.));
270     double angle = ((fabs(ey - sy) < 1e-9) ? 1 : ((ey - sy) /
271 → fabs(ey - sy))) * acos((ex - sx) / norm) * 180. / M_PI;
272     tx = sx;
273     ty = sy;
274     tangle = angle;
275     courbe_hilbert_rule_L(ecran, norm / pow(2., n - 1), n, r, g,
276 → b, a);
277 }
278
279 void tapis_sierpinski_aux(SDL_Surface * ecran, double sx, double
280 → sy, double ex, double ey, int n, int r, int g, int b, int a) {
281     int nx = (ey - sy);
282     int ny = -(ex - sx);
283     if(n <= 1) {
284         return;
285     }
286     int i, j;
287     for(i = 0; i < 3; ++i) {
288         for(j = 0; j < 3; ++j) {
289             if(i == 1 && j == 1) continue;
290             tapis_sierpinski_aux(ecran,
291                 sx + ((i) / 3.) * (ex - sx) + ((j) / 3.) * nx, sy + ((i)
292 → / 3.) * (ey - sy) + ((j) / 3.) * ny,
293                 sx + ((i + 1) / 3.) * (ex - sx) + ((j) / 3.) * nx, sy +
294 → ((i + 1) / 3.) * (ey - sy) + ((j) / 3.) * ny,
295                 n - 1, r, g, b, a);
296         }
297     }
298     Sint16 xs[] = {
299         sx + (1. / 3.) * (ex - sx) + (1. / 3.) * nx,
300         sx + (1. / 3.) * (ex - sx) + (2. / 3.) * nx,
301         sx + (2. / 3.) * (ex - sx) + (2. / 3.) * nx,
302         sx + (2. / 3.) * (ex - sx) + (1. / 3.) * nx
303     };
304 }
```

```

298     Sint16 ys[] = {
299         sy + (1. / 3.) * (ey - sy) + (1. / 3.) * ny,
300         sy + (1. / 3.) * (ey - sy) + (2. / 3.) * ny,
301         sy + (2. / 3.) * (ey - sy) + (2. / 3.) * ny,
302         sy + (2. / 3.) * (ey - sy) + (1. / 3.) * ny
303     };
304     filledPolygonRGBA(ecran, xs, ys, 4, 0.9 * r, 0.9 * g, 0.9 * b,
305     → a);
306 }
307
308 void tapis_sierpinski(SDL_Surface * ecran, double sx, double sy,
309 → double ex, double ey, int n, int r, int g, int b, int a) {
310     int nx = (ey - sy);
311     int ny = -(ex - sx);
312     Sint16 xs[] = {
313         sx,
314         sx + nx,
315         ex + nx,
316         ex
317     };
318     Sint16 ys[] = {
319         sy,
320         sy + ny,
321         ey + ny,
322         ey
323     };
324     filledPolygonRGBA(ecran, xs, ys, 4, r / 10, g / 10, b / 10,
325     → a);
326     tapis_sierpinski_aux(ecran, sx, sy, ex, ey, n, r, g, b, a);
327 }
328
329 void triangle_sierpinski_aux(SDL_Surface * ecran, double sx,
330 → double sy, double ex, double ey, int n, int r, int g, int b,
331 → int a) {
332     int nx = (ey - sy);
333     int ny = -(ex - sx);
334     if(n <= 1) {
335         return;
336     }
337     triangle_sierpinski_aux(ecran, sx, sy, 0.5 * (sx + ex), 0.5 *
338     → (sy + ey), n - 1, r, g, b, a);

```

```
333     triangle_sierpinski_aux(ecran, 0.5 * (sx + ex), 0.5 * (sy +
334         ↳ ey), ex, ey, n - 1, r, g, b, a);
335     triangle_sierpinski_aux(ecran,
336         0.25 * (3 * sx + ex) + 0.87 * 0.5 * nx, 0.25 * (3 * sy + ey)
337         ↳ + 0.87 * 0.5 * ny,
338         0.25 * (sx + 3 * ex) + 0.87 * 0.5 * nx, 0.25 * (sy + 3 * ey)
339         ↳ + 0.87 * 0.5 * ny,
340         n - 1, r, g, b, a);
341     Sint16 xs[] = {
342         0.25 * (3 * sx + ex) + 0.87 * 0.5 * nx,
343         0.25 * (sx + 3 * ex) + 0.87 * 0.5 * nx,
344         0.5 * (sx + ex)
345     };
346     Sint16 ys[] = {
347         0.25 * (3 * sy + ey) + 0.87 * 0.5 * ny,
348         0.25 * (sy + 3 * ey) + 0.87 * 0.5 * ny,
349         0.5 * (sy + ey)
350     };
351     filledPolygonRGBA(ecran, xs, ys, 3, r, g, b, a);
352 }
353
354 void triangle_sierpinski(SDL_Surface * ecran, double sx, double
355     ↳ sy, double ex, double ey, int n, int r, int g, int b, int a) {
356     int nx = (ey - sy);
357     int ny = -(ex - sx);
358     Sint16 xs[] = {
359         sx,
360         0.5 * (sx + ex) + 0.87 * nx,
361         ex
362     };
363     Sint16 ys[] = {
364         sy,
365         0.5 * (sy + ey) + 0.87 * ny,
366         ey
367     };
368     filledPolygonRGBA(ecran, xs, ys, 3, r / 10, g / 10, b / 10,
369         ↳ a);
370     triangle_sierpinski_aux(ecran, sx, sy, ex, ey, n, r, g, b, a);
371 }
372
373 const char * fonction_name(int id) {
```

```

369 switch(id) {
370     case 1 : return "Courbe de Koch";
371     case 2 : return "Fractale Cesaro";
372     case 3 : return "Courbe de Koch alternee";
373     case 4 : return "Courbe du Dragon";
374     case 5 : return "Courbe de Gosper";
375     case 6 : return "Courbe de Hilbert";
376     case 7 : return "Tapis de Sierpinski";
377     case 8 : return "Triangle de Sierpinski";
378     default : return "Flocon de Koch";
379 }
380 }
381
382 void (*fonction(int id))(SDL_Surface * ecran, double sx, double
→ sy, double ex, double ey, int n, int r, int g, int b, int a) {
383     switch(id) {
384         case 1 : return courbe_koch;
385         case 2 : return courbe_cesaro;
386         case 3 : return courbe_koch_alt;
387         case 4 : return courbe_dragon;
388         case 5 : return courbe_gosper;
389         case 6 : return courbe_hilbert;
390         case 7 : return tapis_sierpinski;
391         case 8 : return triangle_sierpinski;
392         default : return flocon_koch;
393     }
394 }
395
396 int main() {
397     srand(time(NULL));
398     /* Création d'une fenêtre SDL : */
399     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
400         fprintf(stderr, "Error in SDL_Init : %s\n",
401             SDL_GetError());
402         exit(EXIT_FAILURE);
403     }
404     SDL_Surface * ecran = NULL;
405     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
→ | SDL_DOUBLEBUF)) == NULL) {
406         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
407             SDL_GetError());
408         SDL_Quit();

```

```
407         exit(EXIT_FAILURE);
408     }
409     SDL_WM_SetCaption(titre, NULL);
410
411     /* Placement du joueur au centre de la fenêtre : */
412
413     cx = 0;
414     cy = 0;
415     cz = 10;
416
417     int active = 1;
418     SDL_Event event;
419     int grab = 0;
420     int refresh = 1;
421
422     while(active) {
423
424         if(refresh) {
425             affichage(ecran);
426             SDL_Flip(ecran);
427             refresh = 0;
428         }
429
430         while(SDL_PollEvent(&event)) {
431
432             switch(event.type) {
433                 /* Utilisateur clique sur la croix de la fenêtre : */
434                 case SDL_QUIT : {
435                     active = 0;
436                 } break;
437
438                 /* Utilisateur enfonce une touche du clavier : */
439                 case SDL_KEYDOWN : {
440                     switch(event.key.keysym.sym) {
441                         /* Touche Echap : */
442                         case SDLK_ESCAPE : {
443                             active = 0;
444                         } break;
445                     }
446                 } break;
447 }
```

```
448     case SDL_KEYUP : {
449         switch(event.key.keysym.sym) {
450             case SDLK_UP : {
451                 current_function = (current_function + 1) %
452                     functions_count;
453                 refresh = 1;
454             } break;
455
456             case SDLK_DOWN : {
457                 current_function = (current_function +
458                     → functions_count - 1) % functions_count;
459                 refresh = 1;
460             } break;
461
462             case SDLK_RIGHT : {
463                 iterations++;
464                 refresh = 1;
465             } break;
466
467             case SDLK_LEFT : {
468                 iterations--;
469                 refresh = 1;
470             } break;
471
472             case SDLK_a : {
473                 segments++;
474                 refresh = 1;
475             } break;
476
477             case SDLK_q : {
478                 segments--;
479                 refresh = 1;
480             } break;
481         }
482     } break;
483
484     case SDL_MOUSEBUTTONDOWN : {
485         switch(event.button.button) {
486             case SDL_BUTTON_WHEELUP : {
487                 cz *= 0.8;
488                 if(cz < 1e-9) {
489
```

```
487         cz = 1e-9;
488     }
489     refresh = 1;
490 } break;

491
492 case SDL_BUTTON_WHEELDOWN : {
493     cz /= 0.8;
494     if(cz > 1e9) {
495         cz = 1e9;
496     }
497     refresh = 1;
498 } break;

499
500 case SDL_BUTTON_LEFT : {
501     grab = 1;
502 } break;
503 }
504 } break;

505
506 case SDL_MOUSEBUTTONDOWN : {
507     switch(event.button.button) {
508         case SDL_BUTTON_LEFT : {
509             grab = 0;
510         } break;
511     }
512 } break;

513
514 case SDL_MOUSEMOTION : {
515     if(grab) {
516         cx += camera_depuis_ecran_z(-event.motion.xrel);
517         cy += camera_depuis_ecran_z(-event.motion.yrel);
518         refresh = 1;
519     }
520 } break;
521 }
522 }

523     SDL_Delay(1000 / 60);
524 }

525     SDL_FreeSurface(ecran);
```

```
528     SDL_Quit();  
529     exit(EXIT_SUCCESS);  
530 }
```

A.7 Tableaux

A.7.1 Échauffement

Correction 59 (★☆ Création et affectation).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void afficher_tableau(int tableau[]) {
5     int i;
6     printf("[");
7     for(i = 0; tableau[i] >= 0; ++i) {
8         if(i) printf(", ");
9         printf("%d", tableau[i]);
10    }
11    printf("]\n");
12 }
13
14 int main() {
15     int tableau[] = {1, 2, 3, 4, 5, -1};
16     int indice;
17     int valeur;
18     afficher_tableau(tableau);
19     printf("Quel indice affecter ? ");
20     scanf("%d", &indice);
21     printf("Pour quelle valeur ? ");
22     scanf("%d", &valeur);
23     tableau[indice] = valeur;
24     afficher_tableau(tableau);
25     exit(EXIT_SUCCESS);
26 }
```

Correction 60 (★☆ Lire valeurs).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void afficher_tableau(int tableau[], int taille) {
```

```

5   int i;
6   printf("[");
7   for(i = 0; i < taille; ++i) {
8     if(i) printf(", ");
9     printf("%d", tableau[i]);
10  }
11  printf("]\n");
12 }

13
14 int main() {
15   const int taille = 5;
16   int tableau[taille];
17   int i;
18   printf("Saisir %d valeurs : ", taille);
19   for(i = 0; i < taille; ++i) {
20     scanf("%d", &tableau[i]);
21   }
22   afficher_tableau(tableau, taille);
23   exit(EXIT_SUCCESS);
24 }
```

A.7.2 Entrainement

Correction 61 (★★ Statistiques sur un tableau).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define TAILLE 10
5 /* min_index renvoie l'indice du plus petit élément de values */
6 int min_index(int values[], int taille);
7 /* min_value renvoie le plus petit élément de values */
8 int min_value(int values[], int taille);
9 /* max_index renvoie l'indice du plus grand élément de values */
10 int max_index(int values[], int taille);
11 /* max_value renvoie le plus grand élément de values */
12 int max_value(int values[], int taille);
13 /* moyenne renvoie la moyenne des éléments de values */
14 float moyenne(int values[], int taille);
```

```
15
16 int main() {
17     int valeurs[TAILLE] = {5, 9, 1, 4, 8, 3, 0, 6, 2, 7};
18     printf("minimum : valeurs[%d] = %d\n",
19            min_index(valeurs, TAILLE), min_value(valeurs, TAILLE));
20     printf("maximum : valeurs[%d] = %d\n",
21            max_index(valeurs, TAILLE), max_value(valeurs, TAILLE));
22     printf("moyenne : %g\n", moyenne(valeurs, TAILLE));
23     exit(EXIT_SUCCESS);
24 }
25
26 int min_index(int values[], int taille) {
27     int index = 0;
28     int i;
29     for(i = 1; i < taille; ++i) {
30         if(values[i] < values[index]) {
31             index = i;
32         }
33     }
34     return index;
35 }
36
37 int min_value(int values[], int taille) {
38     return values[min_index(values, taille)];
39 }
40
41 int max_index(int values[], int taille) {
42     int index = 0;
43     int i;
44     for(i = 1; i < taille; ++i) {
45         if(values[i] > values[index]) {
46             index = i;
47         }
48     }
49     return index;
50 }
51
52 int max_value(int values[], int taille) {
53     return values[max_index(values, taille)];
54 }
```

```

56 float moyenne(int values[], int taille) {
57     long somme = 0;
58     int i;
59     for(i = 0; i < taille; ++i) {
60         somme += values[i];
61     }
62     return (float)somme / taille;
63 }
```

Correction 62 (★★ Coder les fonction de string.h).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int esgi_strlen(const char texte[]);
5 void esgi_strcpy(char destination[], const char source[]);
6 void esgi_strcat(char destination[], const char source[]);
7 int esgi_strcmp(const char first[], const char second[]);
8
9 int main() {
10     char texte[] = "Welcome to ESGI !";
11     char hello[] = "Hello";
12     char copie[50];
13     printf("esgi_strlen(\"%s\") = %d\n", texte, esgi_strlen(texte));
14     esgi_strcpy(copie, "Eleve, ");
15     printf("copie = \"%s\"\n", copie);
16     esgi_strcat(copie, texte);
17     printf("copie = \"%s\"\n", copie);
18     printf("esgi_strcmp(\"Hello\", \"Hello\") = %d = 0\n",
19            esgi_strcmp(hello, "Hello"));
20     printf("esgi_strcmp(\"Hello\", \"Bonjour\") = %d > 0\n",
21            esgi_strcmp(hello, "Bonjour"));
22     printf("esgi_strcmp(\"Hello\", \"Hell\") = %d > 0\n",
23            esgi_strcmp(hello, "Hell"));
24     printf("esgi_strcmp(\"Bonjour\", \"Hello\") = %d < 0\n",
25            esgi_strcmp("Bonjour", hello));
26     exit(EXIT_SUCCESS);
27 }
28
29 int esgi_strlen(const char texte[]) {
```

```
26 int i;
27 /* Nous parcourons la chaîne de caractères jusqu'au marqueur */
28 /* de fin. */ *
29 for(i = 0; texte[i] != '\0'; ++i) {
30 }
31 return i;
32 }

33 void esgi_strcpy(char destination[], const char source[]) {
34     int i;
35     /* Nous copions individuellement les caractères jusqu'à */
36     /* atteindre le marqueur de fin. */ *
37     for(i = 0; source[i] != '\0'; ++i) {
38         destination[i] = source[i];
39     }
40     /* Nous pensons à l'ajouter en fin de la chaîne. */
41     destination[i] = '\0';
42 }
43

44 void esgi_strcat(char destination[], const char source[]) {
45     int i;
46     /* Nous recherchons la position du marqueur de fin de la */
47     /* chaîne destination. */ *
48     int offset = esgi_strlen(destination);
49     /* Nous copions les caractères de la chaîne source */
50     /* relativement à la fin de destination. */ *
51     for(i = 0; source[i] != '\0'; ++i) {
52         destination[i + offset] = source[i];
53     }
54     destination[i + offset] = '\0';
55 }
56

57 int esgi_strcmp(const char first[], const char second[]) {
58     int i;
59     /* Tant que les deux chaînes fournissent des caractères */
60     for(i = 0; first[i] != '\0' && second[i] != '\0'; ++i) {
61         /* S'ils sont différents, nous déterminons la précédence */
62         /* lexicographique entre les chaînes. */ *
63         if(first[i] < second[i]) {
64             return -1;
65         } else if(first[i] > second[i]) {
```

```

67     return 1;
68 }
69 /* Sinon les caractères sont identiques. */
70 }
71 /* Nous gérons le cas où une chaîne serait préfixe de      */
72 /* l'autre.                                                 */
73 if(first[i] < second[i]) {
74     return -1;
75 } else if(first[i] > second[i]) {
76     return 1;
77 } else {
78     return 0;
79 }
80 }
```

Correction 63 (★★★ Compter les occurrences de lettres).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     char buffer[1024];
6     printf("Entrez du texte : ");
7     scanf("%[^\\n]", buffer);
8     printf("Texte : \"%s\"\n", buffer);
9     int count;
10    char c;
11    int i;
12    for(c = 'a'; c <= 'z'; ++c) {
13        count = 0;
14        for(i = 0; buffer[i] != '\\0'; ++i) {
15            if(buffer[i] == c || buffer[i] == (c + 'A' - 'a')) {
16                ++count;
17            }
18        }
19        if(count > 0) {
20            printf("\'%c\' : %d\n", c, count);
21        }
22    }
23    exit(EXIT_SUCCESS);
}
```

24 }

Correction 64 (★★★ Décodage Vigenère).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void vigenere(char texte[], char cle[], int sign, char message[])
6 {
7     int i, j;
8     int value;
9     int offset;
10    for(i = 0, j = 0; texte[i] != '\0'; ++i) {
11        /* Nous déterminons la position dans l'alphabet : */
12        if(texte[i] >= 'A' && texte[i] <= 'Z') {
13            value = texte[i] - 'A';
14            message[i] = 'A';
15        } else if(texte[i] >= 'a' && texte[i] <= 'z') {
16            value = texte[i] - 'a';
17            message[i] = 'a';
18        } else {
19            /* Dans le cas où ce n'est pas une lettre, nous
20             * n'appliquons pas de traitement.
21             */
22            message[i] = texte[i];
23            continue;
24        }
25        /* Nous récupérons le décalage de la clé : */
26        if(cle[j] >= 'A' && cle[j] <= 'Z') {
27            offset = cle[j] - 'A';
28        } else if(cle[j] >= 'a' && cle[j] <= 'z') {
29            offset = cle[j] - 'a';
30        }
31        /* Nous calculons la position décalée de la lettre dans
32         * l'alphabet depuis la clé.
33         */
34        value = ((value + sign * offset) % 26 + 26) % 26;
35        message[i] += value;
36        /* Nous consommons la position utilisée dans la clé. */
37        j = (j + 1) % strlen(cle);
38    }

```

```

36     message[i] = '\0';
37 }
38
39 void lire_texte(char texte[]) {
40     int c;
41     int i;
42     c = getchar();
43     for(i = 0; c != '\n' && c != EOF; ++i) {
44         texte[i] = c;
45         c = getchar();
46     }
47     texte[i] = '\0';
48 }
49
50 int main() {
51     char cle[50];
52     char choix;
53     char texte[500];
54     char message[500];
55     printf("Entrez une clé : ");
56     scanf("%s", cle);
57     do {
58         printf("Que voulez-vous faire ?\nc - coder\n\t- decoder\nq -
59             \t\tquitter\nvotre choix : ");
60         scanf(" %c ", &choix);
61         switch(choix) {
62             case 'c' :
63                 printf("Entrez un message : ");
64                 lire_texte(texte);
65                 printf("texte = \"%s\"\n", texte);
66                 vigenere(texte, cle, 1, message);
67                 printf("codage_vigenere(K = \"%s\", M = \"%s\") =
68                     \n \"%s\"\n", cle, texte, message);
69                 break;
70             case 'd' :
71                 printf("Entrez un message : ");
72                 lire_texte(texte);
73                 vigenere(texte, cle, -1, message);
74                 printf("decodage_vigenere(K = \"%s\", M = \"%s\") =
75                     \n \"%s\"\n", cle, texte, message);
76                 break;
77         }
78     }
79 }
```

```
74     }
75 } while(choix != 'q');
76 exit(EXIT_SUCCESS);
77 }
```

Correction 65 (★★★ Gestion d'une liste d'entiers).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 #define CAPACITE_MAX 1000
7
8 int valeurs[CAPACITE_MAX];
9 int taille;
10
11 void init() {
12     taille = 0;
13 }
14
15 void ajouter(int valeur) {
16     if(taille >= CAPACITE_MAX) {
17         printf("[Info] : taille maximale atteinte.\n");
18         return;
19     }
20     valeurs[taille++] = valeur;
21 }
22
23 void supprimer() {
24     if(taille <= 0) {
25         printf("[Info] : aucune valeur.\n");
26         return;
27     }
28     printf("[Info] : dernière valeur de la liste : %d\n",
29            valeurs[--taille]);
30 }
31
32 void afficher() {
33     printf("Liste : [");
```

```

34 int i;
35 for(i = 0; i < taille; ++i) {
36     if(i) {
37         printf(", ");
38     }
39     printf("%d", valeurs[i]);
40 }
41 printf("]\n");
42 }

43
44 void trier(int vals[], int taille) {
45     int min_cur = 0;
46     int max_cur = taille - 1;
47     int i;
48     /* Tant que les curseurs de fin et de début ne se croisent */
49     /* pas : */
50     while(min_cur < max_cur) {
51         for(i = min_cur; i < max_cur + 1; ++i) {
52             /* Nous échangeons le minimum actuel avec la position */
53             /* regardée si plus petite :
54             if(vals[i] < vals[min_cur]) {
55                 vals[i] ^= vals[min_cur];
56                 vals[min_cur] ^= vals[i];
57                 vals[i] ^= vals[min_cur];
58             }
59             /* Nous échangeons la maximum avec la position actuelle */
60             /* si plus grande :
61             if(vals[i] > vals[max_cur]) {
62                 vals[i] ^= vals[max_cur];
63                 vals[max_cur] ^= vals[i];
64                 vals[i] ^= vals[max_cur];
65             }
66         }
67         /* Nous avançons d'un pas en tête et reculons d'un pas en */
68         /* queue de la liste. */
69         ++min_cur;
70         --max_cur;
71     }
72 }
73
74 void stats() {

```

```
75 int tmp_vals[CAPACITE_MAX];
76 int i;
77 double somme = 0;
78 if(taille <= 0) {
79     printf("[Info] : aucune valeur.\n");
80     return;
81 }
82 /* Nous sommes et sauvegardons les valeurs de la liste      */
83 /* actuelle.                                                 */
84 for(i = 0; i < taille; ++i) {
85     somme += tmp_vals[i] = valeurs[i];
86 }
87 /* Nous trions la liste de valeurs sauvegardées pour      */
88 /* déterminer médianes et quartiles.                         */
89 trier(tmp_vals, taille);
90 double moyenne = somme / taille;
91 somme = 0;
92 /* Nous sommes les carrés des distances à la moyenne pour */
93 /* calculer l'écart type :                                     */
94 for(i = 0; i < taille; ++i) {
95     somme += pow(moyenne - valeurs[i], 2.);
96 }
97 double ecart_type = sqrt(somme / taille);
98 double min, q1, q2, q3, max;
99 /* Nous calculons extremums et quartiles depuis la liste   */
100 /* triée des valeurs :                                         */
101 min = tmp_vals[0];
102 q1 = 0.25 * (tmp_vals[(taille - 1) / 4]
103             + tmp_vals[(taille - 0) / 4]
104             + tmp_vals[(taille + 1) / 4]
105             + tmp_vals[(taille + 2) / 4]);
106 q2 = 0.5 * (tmp_vals[(taille - 1) / 2]
107             + tmp_vals[taille / 2]);
108 q3 = 0.25 * (tmp_vals[(3 * taille - 3) / 4]
109             + tmp_vals[(3 * taille - 2) / 4]
110             + tmp_vals[(3 * taille - 1) / 4]
111             + tmp_vals[(3 * taille) / 4]);
112 max = tmp_vals[taille - 1];
113 printf("Statistiques :\n"
114         " - moyenne :      %g\n"
115         " - ecart type :   %g\n"
```

```

116         " - minimum :      %g\n"
117         " - 1er quartile : %g\n"
118         " - mediane :      %g\n"
119         " - 3eme quartile : %g\n"
120         " - maximum :      %g\n",
121         moyenne,
122         ecart_type,
123         min, q1, q2, q3, max);
124     }
125
126 void rechercher(int valeur) {
127     int i;
128     for(i = 0; i < taille; ++i) {
129         if(valeurs[i] == valeur) {
130             printf("%d trouvé à l'indice %d.\n", valeur, i);
131         }
132     }
133 }
134
135 void do_action(char action[]) {
136     /* Nous référeçons les action possibles via cette fonction */
137     /* et récupérons des paramètres si nécessaire : */
138     if(strcmp(action, "ajouter") == 0) {
139         int v;
140         scanf("%d", &v);
141         ajouter(v);
142     } else if(strcmp(action, "supprimer") == 0) {
143         supprimer();
144     } else if(strcmp(action, "rechercher") == 0) {
145         int v;
146         scanf("%d", &v);
147         rechercher(v);
148     } else if(strcmp(action, "afficher") == 0) {
149         afficher();
150     } else if(strcmp(action, "statistiques") == 0) {
151         stats();
152     } else if(strcmp(action, "trier") == 0) {
153         trier(valeurs, taille);
154     } else if(strcmp(action, "quitter") == 0) {
155     } else {
156         printf("[Info] : Action invalide.\n"

```

```
157         "Actions possibles :\n"
158         " - ajouter [VALEUR]\n"
159         " - supprimer\n"
160         " - rechercher [VALEUR]\n"
161         " - afficher\n"
162         " - statistiques\n"
163         " - trier\n"
164         " - quitter\n");
165     }
166 }
167
168 int main() {
169     char action[50];
170     init();
171     do {
172         printf(">>> ");
173         scanf("%s", action);
174         do_action(action);
175     } while(strcmp(action, "quitter") != 0);
176     exit(EXIT_SUCCESS);
177 }
```

Correction 66 (★★★ Dichotomie dans un tableau).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 char read_command() {
5     char command;
6     printf(">>> ");
7     scanf(" %c", &command);
8     return command;
9 }
10
11 int find(int valeur, int liste[], int taille) {
12     int start = 0;
13     int end = taille;
14     int mid;
15     while(end - start > 0) {
16         mid = (start + end) / 2;
```

```

17     if(liste[mid] < valeur) {
18         start = mid + 1;
19     } else if(liste[mid] > valeur) {
20         end = mid;
21     } else {
22         return mid;
23     }
24 }
25 if(taille > 0 && liste[start] == valeur) {
26     return start;
27 }
28 return -1;
29 }

30
31 int add(int valeur, int liste[], int taille) {
32     int i;
33     for(i = taille; i >= 0 && liste[i - 1] > valeur; --i) {
34         liste[i] = liste[i - 1];
35     }
36     liste[i] = valeur;
37     return taille + 1;
38 }

39
40 void display(int liste[], int taille) {
41     int i;
42     printf("[");
43     for(i = 0; i < taille; ++i) {
44         if(i != 0) {
45             printf(", ");
46         }
47         printf("%d", liste[i]);
48     }
49     printf("]\n");
50 }

51
52 int main() {
53     int liste[1024];
54     int taille = 0;
55     char command;
56     while((command = read_command()) != 'q') {
57         switch(command) {

```

```
58     case '+': {
59         int valeur;
60         scanf("%d", &valeur);
61         taille = add(valeur, liste, taille);
62     } break;
63
64     case '=': {
65         int valeur;
66         scanf("%d", &valeur);
67         if(find(valeur, liste, taille) >= 0) {
68             printf("%d existe dans la liste.\n", valeur);
69         } else {
70             printf("%d non trouvé dans la liste.\n", valeur);
71         }
72     } break;
73
74     case '.': {
75         display(liste, taille);
76     } break;
77
78     default: {
79         printf(
80             "commande non reconnue :\n"
81             "+ [ENTIER] : ajouter une valeur\n"
82             "= [ENTIER] : rechercher une valeur\n"
83             ". : affiche la liste\n"
84             "q : quitter\n"
85         );
86     } break;
87 }
88 }
89 exit(EXIT_SUCCESS);
}
```

Correction 67 (★★★ Suite de Fibonacci).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
```

```

5  /* Fonction récursive naïve pour calculer la suite de */          */
6  /* Fibonacci depuis sa définition : */                                */
7  unsigned long fibo_rec(int n) {
8      if(n <= 1) {
9          return n;
10     }
11     return fibo_rec(n - 1) + fibo_rec(n - 2);
12 }
13
14 /* Version itérative du calcul de la suite de Fibonacci : */
15 unsigned long fibo_lin(int n) {
16     unsigned long u0 = 0, u1 = 1, r;
17     while(n-- > 0) {
18         r = u1;
19         u1 = u1 + u0;
20         u0 = r;
21     }
22     return u0;
23 }
24
25 /* Affectation d'une matrice depuis ses coefficients : */
26 void matrix2x2_set(unsigned long matrix[2][2],
27   unsigned long a, unsigned long b,
28   unsigned long c, unsigned long d) {
29
30     matrix[0][0] = a; matrix[0][1] = b;
31     matrix[1][0] = c; matrix[1][1] = d;
32 }
33
34 /* Affectation d'un vecteur depuis ses coefficients : */
35 void vector2_set(unsigned long vector[2],
36   unsigned long a, unsigned long b) {
37
38     vector[0] = a;
39     vector[1] = b;
40 }
41
42 /* Multiplication matrice * vecteur */
43 void matrix2x2_mul_vector2(
44   unsigned long res[2],
45   unsigned long first[2][2],

```

```
46     unsigned long second[2]) {
47
48     unsigned long x = second[0], y = second[1];
49
50     res[0] = first[0][0] * x + first[0][1] * y;
51     res[1] = first[1][0] * x + first[1][1] * y;
52 }
53
54 /* Copie d'une matrice pour affectation : */
55 void matrix2x2_copy(
56     unsigned long dest[2][2],
57     unsigned long src[2][2]) {
58
59     int i, j;
60     for(i = 0; i < 2; ++i) {
61         for(j = 0; j < 2; ++j) {
62             dest[i][j] = src[i][j];
63         }
64     }
65 }
66
67 /* Multiplication matrice * matrice : */
68 void matrix2x2_mul_matrix2x2(
69     unsigned long res[2][2],
70     unsigned long first[2][2],
71     unsigned long second[2][2]) {
72
73     unsigned long a = first[0][0], b = first[0][1];
74     unsigned long c = first[1][0], d = first[1][1];
75
76     unsigned long x = second[0][0], z = second[0][1];
77     unsigned long y = second[1][0], w = second[1][1];
78
79     res[0][0] = a * x + b * y; res[0][1] = a * z + b * w;
80     res[1][0] = c * x + d * y; res[1][1] = c * z + d * w;
81 }
82
83 void matrix2x2_display(unsigned long matrix[2][2]) {
84     printf("Matrix2x2 :\n");
85     int i, j;
86     for(i = 0; i < 2; ++i) {
```

```

87     if(i != 0) {
88         printf("\n");
89     }
90     for(j = 0; j < 2; ++j) {
91         if(j != 0) {
92             printf(" ");
93         }
94         printf("%18lu", matrix[i][j]);
95     }
96 }
97 }

98 /* Fonction puissance d'une matrice (complexité linéaire) : */
99 void matrix2x2_power_naive(
100     unsigned long res[2][2],
101     unsigned long matrix[2][2],
102     int n) {
103
104     unsigned long tmp[2][2];
105     /* tmp = 1 (identité matrice) */
106     matrix2x2_set(tmp,
107         1, 0,
108         0, 1);
109     while(n-- > 0) {
110         /* tmp = tmp * matrix */
111         matrix2x2_mul_matrix2x2(tmp, tmp, matrix);
112     }
113     /* res = tmp */
114     matrix2x2_copy(res, tmp);
115 }
116 }

117 /* Fonction puissance rapide d'une matrice (complexité
118 /* logarithmique) : */ */
119 void matrix2x2_power_fast(
120     unsigned long res[2][2],
121     unsigned long matrix[2][2],
122     int n) {
123
124     unsigned long tmp[2][2];
125     unsigned long val[2][2];
126     /* tmp = 1 (identité matrice) */
127

```

```
128     matrix2x2_set(tmp,
129         1, 0,
130         0, 1);
131     /* val = matrix */
132     matrix2x2_copy(val, matrix);
133     while(n > 0) {
134         if(n % 2 == 1) {
135             /* tmp = tmp * val */
136             matrix2x2_mul_matrix2x2(tmp, tmp, val);
137         }
138         /* val = val * val */
139         matrix2x2_mul_matrix2x2(val, val, val);
140         n /= 2;
141     }
142     /* res = tmp */
143     matrix2x2_copy(res, tmp);
144 }
145
146 /* Calcul de la suite de Fibonacci depuis la méthode proposée */
147 /* avec une matrice 2x2 : */
148 unsigned long fibo_matrix(int n) {
149     unsigned long matrix[2][2];
150     unsigned long vector[2];
151     matrix2x2_set(matrix,
152         1, 1,
153         1, 0);
154     vector2_set(vector,
155         1, 0);
156     matrix2x2_power_fast(matrix, matrix, n);
157     matrix2x2_mul_vector2(vector, matrix, vector);
158     return vector[1];
159 }
160
161 int main() {
162     int n;
163     printf("Entrez la valeur de l'\indice à calculer : ");
164     scanf("%d", &n);
165     unsigned long res;
166     clock_t start, end;
167     start = clock();
168     res = fibo_matrix(n);
```

```

169 end = clock();
170 printf("fibo_matrix(%d) = %lu (%g s)\n", n, res,
171     (double)(end - start) / CLOCKS_PER_SEC);
172 start = clock();
173 res = fibo_lin(n);
174 end = clock();
175 printf("fibo_lin(%d)      = %lu (%g s)\n", n, res,
176     (double)(end - start) / CLOCKS_PER_SEC);
177 if(n <= 50) {
178     start = clock();
179     res = fibo_rec(n);
180     end = clock();
181     printf("fibo_rec(%d)      = %lu (%g s)\n", n, res,
182         (double)(end - start) / CLOCKS_PER_SEC);
183 } else {
184     printf("fibo_rec(%d)      = ? (trop long)\n", n);
185 }
186 exit(EXIT_SUCCESS);
187 }
```

Correction 68 (★★★ OOO Chiffrement par permutation d'alphabet).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5
6 void init(char alpha[27]) {
7     int i;
8     for(i = 0; i < 26; ++i) {
9         alpha[i] = 'a' + i;
10    }
11    alpha[i] = '\0';
12 }
13
14 void permute(char alpha[27]) {
15     int first, second;
16     char tmp;
17     int i;
18     for(i = 0; i < 1000; ++i) {
```

```
19     first = rand() % 26;
20     second = rand() % 26;
21     tmp = alpha[first];
22     alpha[first] = alpha[second];
23     alpha[second] = tmp;
24 }
25 }

26
27 void inverse(char alpha[27], char ialpha[27]) {
28     int i;
29     for(i = 0; i < 26; ++i) {
30         ialpha[alpha[i] - 'a'] = 'a' + i;
31     }
32 }

33
34 void apply(char alpha[27], char texte[]) {
35     int i;
36     for(i = 0; texte[i] != '\0'; ++i) {
37         if('a' <= texte[i] && texte[i] <= 'z')
38             texte[i] = alpha[texte[i] - 'a'];
39         else if('A' <= texte[i] && texte[i] <= 'Z')
40             texte[i] = alpha[texte[i] - 'A'] + 'A' - 'a';
41     }
42 }

43
44 int main(int argc, char * argv[]) {
45     srand(time(NULL));
46     char balpha[27];
47     char ialpha[27];
48     init(balpha);
49     init(ialpha);
50     if(argc <= 1 || (argc <= 2 && strcmp(argv[1], "decode") == 0)) {
51         printf("Entrez une commande du type :\n"
52               " %s [ACTION] [optionnel : ALPHABET]\n"
53               " Avec [ACTION] :\n"
54               " \"alpha\" : génère une permutation d'alphabet.\n"
55               " \"code\" : encode le texte par l'alphabet.\n"
56               " \"decode\" : décode le texte depuis l'alphabet.\n",
57               argv[0]);
58         exit(EXIT_FAILURE);
59 }
```

```
59     char * alpha = (argc > 2) ? argv[2] : balpha;
60     char * action = argv[1];
61
62     permutate(balpha);
63     inverse(alpha, ialpha);
64     printf("alpha : \"%s\"\n", alpha);
65     printf("ialpha : \"%s\"\n", ialpha);
66
67     if(strcmp(action, "alpha") == 0) {
68         exit(EXIT_SUCCESS);
69     }
70
71     char texte[2048];
72     printf("Saisir une ligne : ");
73     scanf("%[^\\n]", texte);
74     printf("Texte      : \"%s\"\n", texte);
75     if(strcmp(action, "code") == 0) {
76         apply(alpha, texte);
77         printf("Texte codé    : \"%s\"\n", texte);
78     }
79     if(strcmp(action, "decode") == 0) {
80         apply(ialpha, texte);
81         printf("Texte décodé : \"%s\"\n", texte);
82     }
83     exit(EXIT_SUCCESS);
84 }
```

A.8 Pointeurs

A.8.1 Cours

Correction 69 (★★★ Fonction pour échanger des valeurs).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void echanger(int * first, int * second) {
5     int temporaire;
6     temporaire = *first;
7     *first = *second;
8     *second = temporaire;
9 }
10
11 int main() {
12     int first = 42, second = 1337;
13     printf("first = %d, second = %d\n", first, second);
14     echanger(&first, &second);
15     printf("first = %d, second = %d\n", first, second);
16     exit(EXIT_SUCCESS);
17 }
```

A.8.2 Échauffement

Correction 70 (★ Pointeurs sur des variables).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int ma_maison = 42;
6     int nouvelle_maison = 0;
7     int * mon_adresse = &ma_maison;
8     int * nouvelle_adresse = &nouvelle_maison;
9     printf("[%c] Je sais où est ma maison ?\n",
10            (mon_adresse == &ma_maison) ? 'V' : 'X');
11     printf("[%c] Je sais où déménager ?\n",
12            (nouvelle_adresse == &nouvelle_maison) ? 'V' : 'X');
```

```

12     (nouvelle_adresse == &nouvelle_maison) ? 'V' : 'X');
13 *nouvelle_adresse = *mon_adresse;
14 *mon_adresse = 0;
15 printf("[%c] J'ai tout déménagé dans ma nouvelle maison ?\n",
16     (ma_maison == 0) ? 'V' : 'X');
17 printf("[%c] J'ai oublié des choses dans ma maison ?\n",
18     (nouvelle_maison == 42) ? 'V' : 'X');
19 mon_adresse = nouvelle_adresse;
20 (*mon_adresse)++;
21 printf("[%c] J'habite à ma nouvelle adresse ?\n",
22     (mon_adresse == &nouvelle_maison) ? 'V' : 'X');
23 printf("[%c] J'ai ajouté des choses dans ma maison ?\n",
24     (nouvelle_maison == 43) ? 'V' : 'X');
25 exit(EXIT_SUCCESS);
26 }
```

Correction 71 (★★ Pointinception).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int valeur = 42;
6     int * ptr = &valeur;
7     int ** pptr = &ptr;
8     int *** ppptr = &pptr;
9     printf("valeur via valeur : %d\n", valeur);
10    printf("valeur via ptr : %d\n", *ptr);
11    printf("valeur via pptr : %d\n", **pptr);
12    printf("valeur via ppptr : %d\n", ***ppptr);
13    exit(EXIT_SUCCESS);
14 }
```

Correction 72 (★★★ Construction liste extensible).

```

1 /* Correction des fuites de mémoire : */
2 #include <stdio.h>
3 #include <stdlib.h>
```

```
4
5 int main() {
6     const int ajouts = 10000;
7     int * liste = malloc(sizeof(int));
8     int * nouvelle_liste = NULL;
9     int taille = 0;
10    int i;
11    int j;
12    for(i = 0; i < ajouts; ++i) {
13        liste[taille++] = i;
14        nouvelle_liste = malloc(sizeof(int) * (taille + 1));
15        for(j = 0; j < taille; ++j) {
16            nouvelle_liste[j] = liste[j];
17        }
18        /* L'affectation de liste ne libère pas la plage */          */
19        /* précédemment référencée. */                                */
20        free(liste);
21        liste = nouvelle_liste;
22    }
23    printf("liste : [");
24    for(i = 0; i < taille; ++i) {
25        if(i) printf(", ");
26        printf("%d", liste[i]);
27    }
28    printf("]\n");
29    free(liste);
30    exit(EXIT_SUCCESS);
31 }
```

```
1 /* Optimisation et remplacement de code non pertinent : */
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     const int ajouts = 1000000000;
7     int * liste = malloc(sizeof(int));
8     int taille = 0;
9     int i;
10    for(i = 0; i < ajouts; ++i)
11        liste[taille++] = i;
```

```

12     if((liste = (int *)realloc(liste, sizeof(int) * (taille + 1)))
13         == NULL) {
14         printf("Erreur allocation.\n");
15         exit(EXIT_FAILURE);
16     }
17     /* L'affichage n'est plus pertinent ici. */
18     /*printf("liste : [");
19     for(i = 0; i < taille; ++i) {
20         if(i) printf(",");
21         printf("%d", liste[i]);
22     }
23     printf("]\n");*/
24     free(liste);
25     exit(EXIT_SUCCESS);
26 }
```

```

1  /* Optimisation de l'allocation dynamique : */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main() {
6      const int ajouts = 1000000000;
7      int * liste = NULL;
8      int capacite = 0;
9      int taille = 0;
10     int i;
11     for(i = 0; i < ajouts; ++i) {
12         if(taille >= capacite) {
13             capacite = capacite * 2 + 10;
14             if((liste = (int *)realloc(liste, sizeof(int) * capacite)) 
15                 == NULL) {
16                 printf("Erreur allocation.\n");
17                 exit(EXIT_FAILURE);
18             }
19             liste[taille++] = i;
20         }
21     }
22     /*printf("liste : [");
```

```
23 for(i = 0; i < taille; ++i) {
24     if(i) printf(" ");
25     if(i == 5) {
26         printf("... ");
27         i = ajouts - 5;
28         continue;
29     }
30     printf("%d", liste[i]);
31 }
32 printf("]\n");*/
33 free(liste);
34 liste = NULL;
35 exit(EXIT_SUCCESS);
36 }
```

A.8.3 Consolidation

Correction 73 (★★ Échanger dans un tableau).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void echanger(int * first, int * second) {
5     int tmp = *first;
6     *first = *second;
7     *second = tmp;
8 }
9
10 int main() {
11     int tableau[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
12     int indice_first = 2;
13     int indice_second = 4;
14     echanger(tableau + indice_first, tableau + indice_second);
15     int i;
16     printf("tableau : [");
17     for(i = 0; i < 10; ++i) {
18         if(i) printf(" ");
19         printf("%d", *(tableau + i));
20     }
}
```

```

21   printf("]\n");
22   exit(EXIT_SUCCESS);
23 }
```

Correction 74 (★★★ Tableau de variables).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void print_values(int ** variables) {
5     printf("values : [\n");
6     for(; *variables; ++variables) {
7         printf("\t0x%p : %d\n", *variables, **variables);
8     }
9     printf("]\n");
10 }
11
12 int main() {
13     int a = 1;
14     int b = 2;
15     int c = 3;
16     int i = 0;
17     int * variables[] = {
18         &i,
19         &a,
20         &b,
21         &c,
22         NULL
23     };
24     print_values(variables);
25     a = 42;
26     for(i = 0; variables[i] != NULL; ++i) {
27         ++(*(variables[i]));
28     }
29     print_values(variables);
30     /* Pourquoi a vaut 42 ? */
31     /* Lors du parcours de boucle, l'adresse de i a été           */
32     /* incrémentée par ++(*(variables[i])), ceci a fait sauter      */
33     /* le cas i = 1.                                              */
34     exit(EXIT_SUCCESS);

```

35 }

Correction 75 (★★ Adresse sur un entier).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int variable = 42;
6     unsigned long pointeur = (long)&variable;
7     *((int *)pointeur) = 1337;
8     printf("%d = 1337 ?\n", variable);
9     exit(EXIT_SUCCESS);
10 }
```

Correction 76 (★★ Dupliquer chaîne de caractère via arithmétique de pointeurs).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void chaine_double(char * chaine) {
5     char * aux;
6     for(aux = chaine; *aux != '\0'; ++aux)
7         ;
8     char * end = aux;
9     for(; chaine != aux; ++chaine, ++end) {
10         ;
11         *end = *chaine;
12     }
13     *end = '\0';
14 }
15
16 int main() {
17     char chaine[40] = "Hello ! ";
18     printf("x 1 : %s\n", chaine);
19     chaine_double(chaine);
20     printf("x 2 : %s\n", chaine);
21     chaine_double(chaine);
```

```

22     printf("x 4 : %s\n", chaine);
23     exit(EXIT_SUCCESS);
24 }
```

Correction 77 (★ Visualiser les sauts d'octets et interprétation des valeurs).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void read_char(unsigned char * ptr, int taille) {
6     int i;
7     printf("[");
8     for(i = 0; i < taille; ++i, ++ptr) {
9         if(i) printf(", ");
10        printf("0x%x", *ptr);
11    }
12    printf("]\n");
13 }
14
15 void read_int(unsigned int * ptr, int taille) {
16     int i;
17     printf("[");
18     for(i = 0; i < taille; ++i, ++ptr) {
19         if(i) printf(", ");
20         printf("0x%x", *ptr);
21     }
22     printf("]\n");
23 }
24
25 int main() {
26     char chaine[] = "Exemple de texte";
27     int values[] = {0xFEEDCAFE, 0xCODEFOOD};
28     read_char((unsigned char *)chaine, strlen(chaine));
29     read_int((unsigned int *)values, 2);
30     read_int((unsigned int *)chaine, strlen(chaine) / sizeof(int));
31     read_char((unsigned char *)values, 2 * sizeof(int));
32     exit(EXIT_SUCCESS);
33 }
```

A.8.4 Entrainement

Correction 78 (★★ Concept de pointeur).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define NB_MAISONS 7
5
6 void afficher_maison(int * maison) {
7     /* Affiche l'adresse pointée par maison et le contenu      */
8     /* référencé                                                 */
9     printf("%p -> %d\n", maison, *maison);
10 }
11
12 void afficher_maisons(int * maisons, int nombre) {
13     int i;
14     for(i = 0; i < nombre; ++i) {
15         printf("[%d] : ", i);
16         afficher_maison(maisons + i);
17     }
18 }
19
20 int * adresse_maison(int * maisons, int numero) {
21     /* Utilise l'arithmétique des pointeurs pour obtenir      */
22     /* l'adresse à l'indice numéro depuis l'adresse donnée par */
23     /* maison :                                                 */
24     return maisons + numero;
25 }
26
27 int vider_maison(int * maison) {
28     /* Lecture du contenu référencé par maison */
29     int v = *maison;
30     /* Écriture du contenu référencé par maison */
31     *maison = 0;
32     return v;
33 }
34
35 void vider_camion(int * camion, int * maison) {
36     /* Opération sur les contenus référencés par maison et      */
37     /* camion :                                                 */
38 }
```

```

38     *maison += *camion;
39     *camion = 0;
40 }
41
42 int main() {
43     int * demenageur = NULL;
44     int camion = 0;
45     int maisons[NB_MAISONS];
46     int i;
47
48     for(i = 0; i < NB_MAISONS; ++i) {
49         maisons[i] = i + 1;
50     }
51     afficher_maisons(maisons, NB_MAISONS);
52
53     printf("camion -> %d\n", camion);
54
55     demenageur = adresse_maison(maisons, 3);
56     afficher_maison(demenageur);
57     camion = vider_maison(demenageur);
58     afficher_maison(demenageur);
59
60     printf("camion -> %d\n", camion);
61
62     demenageur = adresse_maison(maisons, 5);
63     afficher_maison(demenageur);
64     vider_camion(&camion, demenageur);
65     afficher_maison(demenageur);
66
67     printf("camion -> %d\n", camion);
68     afficher_maisons(maisons, NB_MAISONS);
69
70     exit(EXIT_SUCCESS);
71 }
```

Correction 79 (★★★ Concaténation de chaînes de caractères).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
```

```
4
5 int main() {
6     char nom[100], prenom[50];
7     char * full_name = NULL;
8     printf("Bonjour, entrez votre nom et prénom : ");
9     scanf("%s %s", nom, prenom);
10
11    /* full_name référence une adresse, ceci induit la nécessité */
12    /* de pointer sur des données valides. */
13    /* Le choix est d'allouer dynamiquement la plage mémoire qui */
14    /* pourra accueillir la concaténation des chaînes. */
15
16    /* Il faut la taille des chaînes et la place pour l'espace */
17    /* et marqueur de fin : */
18    int size = 2 + strlen(nom) + strlen(prenom);
19    /* Allocation dynmatique propre de la plage mémoire */
20    /* demandée : */
21    if((full_name = (char *)malloc(sizeof(char) * size)) == NULL) {
22        printf("Allocation impossible.\n");
23        exit(EXIT_FAILURE);
24    }
25
26    /* Opérations sur la chaîne de caractère via opérations de */
27    /* string.h (full_name est une adresse, les opérations */
28    /* d'affectation et arithmétiques joueraient sur l'adresse */
29    /* et non le contenu référencé) */
30    strcpy(full_name, prenom);
31    strcat(full_name, " ");
32    strcat(full_name, nom);
33
34    printf("Vous êtes donc %s !\n", full_name);
35
36    /* Libération de la plage mémoire demandée : */
37    free(full_name);
38    /* L'adresse n'étant plus valide, elle n'a plus à être */
39    /* connue. */
40    full_name = NULL;
41    exit(EXIT_SUCCESS);
42 }
```

Correction 80 (★★★ Gestion d'une liste de noms).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 char ** mots = NULL;
6 int taille = 0;
7 int capacite = 0;
8
9 /* Mise à jour de la capacité du tableau : */
10 void update() {
11     if(taille < capacite) {
12         return ;
13     }
14     capacite = taille * 2 + 10;
15     if((mots = (char **)realloc(mots, capacite * sizeof(char *))) ==
16     → NULL) {
17         printf("Allocation impossible.\n");
18         exit(EXIT_SUCCESS);
19     }
20 }
21
22 int rechercher(const char * mot) {
23     int i;
24     for(i = 0; i < taille; ++i) {
25         if(strcmp(mots[i], mot) == 0) {
26             return i;
27         }
28     }
29     return -1;
30 }
31
32 int ajouter(const char * mot) {
33     update();
34     if(rechercher(mot) >= 0) {
35         return 0;
36     }
37     mots[taille++] = strdup(mot);
38     return 1;
39 }
```

```
40 int supprimer(const char * mot) {
41     int i;
42     if((i = rechercher(mot)) < 0) {
43         return 0;
44     }
45     free(mots[i]);
46     for( ; i < taille - 1; ++i) {
47         mots[i] = mots[i + 1];
48     }
49     --taille;
50     return 1;
51 }
52
53 void afficher() {
54     printf("Mots : {\n");
55     int i;
56     for(i = 0; i < taille; ++i) {
57         printf(" - \"%s\"\n", mots[i]);
58     }
59     printf("}\n");
60 }
61
62 void do_action(const char * action) {
63     char buffer[100];
64     if(strcmp(action, "ajouter") == 0) {
65         scanf("%s", buffer);
66         if(ajouter(buffer)) {
67             printf("[Info] : ajout réussi.\n");
68         } else {
69             printf("[Info] : %s existe déjà.\n", buffer);
70         }
71     } else if(strcmp(action, "supprimer") == 0) {
72         scanf("%s", buffer);
73         if(supprimer(buffer)) {
74             printf("[Info] : suppression réussie.\n");
75         } else {
76             printf("[Info] : %s n'existe pas dans la liste.\n",
77                   buffer);
78         }
79     } else if(strcmp(action, "rechercher") == 0) {
80         scanf("%s", buffer);
```

```

81     if(rechercher(buffer) >= 0) {
82         printf("[Info] : %s trouvé.\n", buffer);
83     } else {
84         printf("[Info] : %s non trouvé.\n", buffer);
85     }
86 } else if(strcmp(action, "afficher") == 0) {
87     afficher();
88 } else if(strcmp(action, "quitter") == 0) {
89
90 } else {
91     printf("[Info] : \"%s\" action inconnue.\n", action);
92     printf("Actions possibles :\n"
93             " - ajouter [mot]\n"
94             " - supprimer [mot]\n"
95             " - rechercher [mot]\n"
96             " - afficher\n"
97             " - quitter\n");
98 }
99
100
101 int main() {
102     char buffer[100];
103     do {
104         printf(">>> ");
105         scanf("%s", buffer);
106         do_action(buffer);
107     } while(strcmp(buffer, "quitter") != 0);
108     for(--taille; taille >= 0; --taille) {
109         free(mots[taille]);
110     }
111     if(mots) free(mots);
112     exit(EXIT_SUCCESS);
113 }
```

Correction 81 (★★★ Tableau dynamique à deux dimensions).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int ** allouer_grille(int largeur, int hauteur) {
```

```
5     int ** grille = NULL;
6     int * data = NULL;
7
8     /* Allocation d'une plage mémoire pour la première dimension */
9     /* (adresses) et Allocation de la plage mémoire des valeurs */
10    if((grille = (int **)malloc(largeur * sizeof(int *))) == NULL
11    || (data = (int *)calloc(largeur * hauteur, sizeof(int))) ==
12        NULL) {
13        printf("Erreur d'allocation.\n");
14        exit(EXIT_FAILURE);
15    }
16
17    int i;
18    /* Les adresses de la première dimension pointent sur un      */
19    /* emplacement de la plage mémoire des valeurs.                  */
20    /* Cette proposition permet de n'avoir que deux allocation      */
21    /* au lieu de (largeur + 1) allocations et conserver           */
22    /* l'utilisation en tableau à deux dimensions.                  */
23    for(i = 0; i < largeur; ++i) {
24        grille[i] = data + i * hauteur;
25    }
26
27    return grille;
28}
29
30 void afficher(int ** grille, int largeur, int hauteur) {
31     int x, y;
32     for(y = 0; y < hauteur; ++y) {
33         for(x = 0; x < largeur; ++x) {
34             printf("%c", grille[x][y] ? '#' : '-');
35         }
36         printf("\n");
37     }
38
39 void fill(int ** grille, int largeur, int hauteur) {
40     int i;
41     int m = (largeur < hauteur) ? largeur : hauteur;
42     /* Remplissage des diagonales carrés : */
43     for(i = 0; i < (m + 1) / 2; ++i) {
44         grille[i][i] = 1;
```

```

45     grille[i][hauteur - i - 1] = 1;
46     grille[largeur - i - 1][i] = 1;
47     grille[largeur - i - 1][hauteur - i - 1] = 1;
48 }
49 int u = (largeur > hauteur) ? largeur : hauteur;
50 /* finition de la jointure entre les diagonales carrés : */
51 for(i = 0; i < u - m; ++i) {
52     grille[m / 2 + i * (largeur / u)][m / 2 + i * (hauteur / u)]
53         = 1;
54 }
55 }
56
57 void free_grille(int *** grille) {
58     /* Libération de la plage de valeurs : */
59     free(**grille);
60     /* Libération de la plage d'adresses : */
61     free(*grille);
62     /* Perte du lien vers l'adresse de données libérées : */
63     *grille = NULL;
64 }
65
66 int main() {
67     int largeur, hauteur;
68     int ** grille = NULL;
69
70     printf("Entrez largeur et hauteur de la grille : ");
71     scanf("%d %d", &largeur, &hauteur);
72
73     grille = allouer_grille(largeur, hauteur);
74     fill(grille, largeur, hauteur);
75     afficher(grille, largeur, hauteur);
76     free_grille(&grille);
77     exit(EXIT_SUCCESS);
78 }
```

Correction 82 (∞ Labyrinthe).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
```

```
4 #include <time.h>
5 #include <assert.h>
6 #include <SDL/SDL.h>
7 #include <SDL/SDL_gfxPrimitives.h>
8
9 /* Paramètres de la fenêtre : */
10 const int largeur = 800;
11 const int hauteur = 800;
12 const char * titre = "ESGI labyrinthe";
13
14 /* Position de la caméra et échelle de la vue : */
15 float cx, cy, cz;
16
17 int player_x, player_y;
18 int exit_x, exit_y;
19
20 SDL_Surface * ecran = NULL;
21
22 char ** grille = NULL;
23 int grille_largeur = 10;
24 int grille_hauteur = 10;
25
26 double ecran_depuis_camera_x(double x) {
27     return ((x - cx) * largeur / 2) / cz + largeur / 2;
28 }
29
30 double ecran_depuis_camera_y(double y) {
31     return ((y - cy) * hauteur / 2) / cz + hauteur / 2;
32 }
33
34 double ecran_depuis_camera_z(double z) {
35     return (z * largeur / 2.) / cz;
36 }
37
38 double camera_depuis_ecran_x(double x) {
39     return 2. * ((x - largeur / 2) * cz) / largeur + cx;
40 }
41
42 double camera_depuis_ecran_y(double y) {
43     return 2. * ((y - hauteur / 2) * cz) / hauteur + cy;
44 }
```

```

45
46 double camera_depuis_ecran_z(double z) {
47     return (2 * z * cz) / largeur;
48 }
49
50 /* Allocation dynamique d'un tableau à deux dimensions : */
51 void grille_creer(int width, int height) {
52     char * data = NULL;
53     if(grille) {
54         free(*grille);
55         free(grille);
56         grille = NULL;
57     }
58     assert((data = (char *)calloc(width * height, sizeof(char))) !=
59            → NULL);
60     assert((grille = (char **)malloc(width * sizeof(char *))) !=
61            → NULL);
62     int i;
63     for(i = 0; i < width; ++i) {
64         grille[i] = data + (i * height);
65     }
66     grille_largeur = width;
67     grille_hauteur = height;
68 }
69
70 int in_grille(int x, int y) {
71     return x >= 0 && y >= 0 && x < grille_largeur && y <
72           → grille_hauteur;
73 }
74
75 int in_inner_grille(int x, int y) {
76     return x >= 1 && y >= 1 && x < grille_largeur - 1 && y <
77           → grille_hauteur - 1;
78 }
79
80 /* Mise en place d'une pile pour dérecursiver la fonction creuser
81    → : */
82 void stack_update(unsigned long long ** stack, int * taille, int *
83                  → capacite) {
84     if(*taille < *capacite) {
85         if(*taille * 4 < *capacite) {

```

```
80         *capacite = *capacite / 2;
81     } else {
82         return ;
83     }
84 } else {
85     *capacite = *taille * 2 + 10;
86 }
87 assert((*stack = (unsigned long long *)realloc(*stack, *capacite
88     ↪ * sizeof(unsigned long long))) != NULL);
89 }

90 void stack_push(unsigned long long value, unsigned long long **
91     ↪ stack, int * taille, int * capacite) {
92     stack_update(stack, taille, capacite);
93     (*stack)[(*taille)++] = value;
94 }
95
96 int stack_pop(unsigned long long * value, unsigned long long **
97     ↪ stack, int * taille, int * capacite) {
98     if(*taille <= 0) return 0;
99     stack_update(stack, taille, capacite);
100    *value = (*stack)[--(*taille)];
101    return 1;
102 }
103
104 /* Conversion des valeurs manipulées par la pile en coordonnées :
105    */
106 void stack_push_coords(int x, int y, unsigned long long ** stack,
107     ↪ int * taille, int * capacite) {
108     unsigned long long v = (unsigned long long)x * 0x100000000 + y;
109     stack_push(v, stack, taille, capacite);
110 }
111
112 int stack_pop_coords(int * x, int * y, unsigned long long **
113     ↪ stack, int * taille, int * capacite) {
114     unsigned long long v;
115     if(! stack_pop(&v, stack, taille, capacite)) return 0;
116     *x = v / 0x100000000;
117     *y = v % 0x100000000;
118     return 1;
119 }
```

```

115
116 /* Récupère une valeur dans la grille ou une valeur extérieure
117 ← aléatoire : */
118 int get_grille(int x, int y) {
119     if(in_grille(x, y)) return grille[x][y];
120     double v = cos(512 * (y + 0.1 * x + 200) * (0.25 * x + y) * (50
121     ← - 0.2 * y + 0.6 * x));
122     return (v > 0.) ? '#' : '.';
123 }
124
125 int can_move(int x, int y) {
126     return get_grille(x, y) != '#';
127 }
128
129 /* Version dérécursive de creuser : */
130 void creuser_stack(unsigned long long ** stack, int * taille, int
131 ← * capacite) {
132     int x, y;
133     while(stack_pop_coords(&x, &y, stack, taille, capacite)) {
134         if(! in_grille(x, y)) continue ;
135         if(grille[x][y] != '#') continue ;
136         int link4 = 0;
137         int link8 = 0;
138         int i, j;
139         int dx = 1, dy = 0;
140         int tmp;
141         for(i = 0; i < 4; ++i) {
142             link4 += (get_grille(x + dx, y + dy) == '.');
143             tmp = dx;
144             dx = dy;
145             dy = -tmp;
146         }
147
148         /* S'il y a déjà deux liens en connectivité 4, on ignore. */
149         if(link4 > 1) continue ;
150
151         dx = 1; dy = 1;
152         for(i = 0; i < 4; ++i) {
153             link8 += (get_grille(x + dx, y + dy) == '.');
154             tmp = dx;
155             dx = dy;
156         }

```

```
153     dy = -tmp;
154 }
155 link8 += link4;
156
157 if(link8 > 2) continue ;
158
159 if(grille[x][y] != '#') {
160     continue;
161 }
162 grille[x][y] = '.';
163
164 dx = 1; dy = 0;
165 j = rand() % 4;
166 for(i = 0; i < j; ++i) {
167     tmp = dx;
168     dx = dy;
169     dy = -tmp;
170 }
171 for(i = 0; i < 4; ++i) {
172     if(in_grille(x + dx, y + dy) && grille[x + dx][y + dy] ==
173         '#') {
174         stack_push_coords(x + dx, y + dy, stack, taille,
175             capacite);
176     }
177     tmp = dx;
178     dx = dy;
179     dy = -tmp;
180 }
181
182 void placer_sortie() {
183     int x, y;
184     double d;
185     double maxi = 0;
186     int i;
187     /* On sélectionne la sortie la plus éloignée du joueur pour un
188        */
189     /* nombre de proposition en fonction de la taille de la grille.
190        */
191     for(i = 0; i < (grille_largeur + grille_hauteur) / 2; ++i) {
```

```

190     do {
191         x = rand() % grille_largeur;
192         y = rand() % grille_hauteur;
193     } while(grille[x][y] == '#');
194     d = pow(x - player_x, 2.) + pow(y - player_y, 2.);
195     if(i == 0 || d > maxi) {
196         exit_x = x;
197         exit_y = y;
198         maxi = d;
199     }
200 }
201 }
202
203 void grille_generer(int width, int height) {
204     grille_creer(width, height);
205     int x, y;
206     for(x = 0; x < width; ++x) {
207         for(y = 0; y < height; ++y) {
208             grille[x][y] = '#';
209         }
210     }
211     unsigned long long * stack = NULL;
212     int taille = 0;
213     int capacite = 0;
214     stack_push_coords(player_x, player_y, &stack, &taille,
215     ↪ &capacite);
216     creuser_stack(&stack, &taille, &capacite);
217     placer_sortie();
218 }
219
220 int check_finish() {
221     if(! (player_x == exit_x && player_y == exit_y)) return 0;
222     /* Si le joueur a atteint la sortie, on génère le niveau suivant
223     ↪ : */
224     grille_largeur = grille_largeur * 1.1 + 1;
225     grille_hauteur = grille_hauteur * 1.1 + 1;
226     player_x = grille_largeur / 2;
227     player_y = grille_hauteur / 2;
228     clock_t start, end;
229     start = clock();
230     grille_generer(grille_largeur, grille_hauteur);

```

```
229     end = clock();
230     printf("Génération d'une grille %dx%d en %g s\n",
231         → grille_largeur, grille_hauteur, (double)(end - start) /
232         → CLOCKS_PER_SEC);
233     return 1;
234 }
235
236 void init() {
237     player_x = grille_largeur / 2;
238     player_y = grille_hauteur / 2;
239     clock_t start, end;
240     start = clock();
241     grille_generer(grille_largeur, grille_hauteur);
242     end = clock();
243     printf("Génération d'une grille %dx%d en %g s\n",
244         → grille_largeur, grille_hauteur, (double)(end - start) /
245         → CLOCKS_PER_SEC);
246 }
247
248 /* Surcouche de filledPieColor en fonction de la connectivité de
249    la grille : */
250 void part_filledPieColor(SDL_Surface * ecran, int x, int y, int d,
251     → int dx, int dy, int c) {
252     int start_angle = 0, end_angle = 0;
253     switch(dx + 2 * dy) {
254         case 3 : start_angle = 0; end_angle = 90; break;
255         case -1 : start_angle = 270; end_angle = 360; break;
256         case 1 : start_angle = 90; end_angle = 180; break;
257         case -3 : start_angle = 180; end_angle = 270; break;
258         default : return ;
259     }
260     filledPieColor(ecran, x, y, d, start_angle, end_angle, c);
261 }
262
263 /* Fonction de dessin stylisée en fonction de la connectivité :
264    */
265 void draw_box(int x, int y) {
266     int dx = 1, dy = 1;
267     int tmp;
268     int i;
269     int corner, ver, hor;
```

```

263 for(i = 0; i < 4; ++i) {
264     corner = (get_grille(x + dx, y + dy) == '#');
265     hor = (get_grille(x + dx, y) == '#');
266     ver = (get_grille(x, y + dy) == '#');
267     if(hor || ver) {
268         if(hor && ver) {
269
270             boxColor(ecran,
271                 ecran_depuis_camera_x(x), ecran_depuis_camera_y(y),
272                 ecran_depuis_camera_x(x + dx * 0.5),
273                 → ecran_depuis_camera_y(y + dy * 0.5),
274                 0x11417Aff);
275             if(! corner) {
276                 part_filledPieColor(ecran,
277                     ecran_depuis_camera_x(x + dx * 0.5),
278                     → ecran_depuis_camera_y(y + dy * 0.5),
279                     → ecran_depuis_camera_z(0.5),
280                     -dx, -dy,
281                     0x60839Bff);
282             }
283         } else {
284             boxColor(ecran,
285                 ecran_depuis_camera_x(x), ecran_depuis_camera_y(y),
286                 ecran_depuis_camera_x(x + dx * 0.5),
287                 → ecran_depuis_camera_y(y + dy * 0.5),
288                 0x60839Bff);
289         }
290     } else {
291         part_filledPieColor(ecran,
292             ecran_depuis_camera_x(x), ecran_depuis_camera_y(y),
293             → ecran_depuis_camera_z(0.5),
294             dx, dy,
295             0x60839Bff);
296     }
297 }
298 void afficher_grille() {

```

```
299     int x, y;
300     int sx, sy, ex, ey;
301     int step;
302     step = camera_depuis_ecran_z(1);
303     if(step < 1) step = 1;
304     sx = camera_depuis_ecran_x(0) - 0.5 * step;
305     sy = camera_depuis_ecran_y(0) - 0.5 * step;
306     ex = camera_depuis_ecran_x(largeur) + 1.5 * step;
307     ey = camera_depuis_ecran_y(hauteur) + 1.5 * step;
308     /*if(sx < 0) sx = 0;
309     if(sy < 0) sy = 0;
310     if(ex >= grille_largeur) ex = grille_largeur;
311     if(ey >= grille_hauteur) ey = grille_hauteur;*/
312     for(x = sx; x < ex; x += step) {
313         for(y = sy; y < ey; y += step) {
314             switch(get_grille(x, y)) {
315 #ifdef NO_STYLE
316                 case '#': boxColor(ecran,
317                     ecran_depuis_camera_x(x - 0.5), ecran_depuis_camera_y(y
318                         ↵ - 0.5),
319                         ecran_depuis_camera_x(x + 0.5), ecran_depuis_camera_y(y
320                         ↵ + 0.5),
321                         0x11417Aff); break;
322             #else
323                 case '#': draw_box(x, y); break;
324             #endif
325                 default : break;
326             }
327         }
328     }
329
330     void affichage() {
331         /* Remplissage de l'écran par un gris foncé uniforme : */
332         SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 0x9E, 0x7D,
333             ↵ 0x3C));
334         afficher_grille();
335         filledCircleColor(ecran,
336             ecran_depuis_camera_x(player_x),
337             ↵ ecran_depuis_camera_y(player_y),
338             ecran_depuis_camera_z(0.5),
```

```
336     0x66CE48ff);
337     filledCircleColor(ecran,
338         ecran_depuis_camera_x(exit_x), ecran_depuis_camera_y(exit_y),
339         ecran_depuis_camera_z(0.5),
340         0xFDCB22ff);
341 }
342
343 int main(int argc, char * argv[]) {
344     if(argc >= 3) {
345         grille_largeur = atoi(argv[1]);
346         grille_hauteur = atoi(argv[2]);
347     }
348     srand(time(NULL));
349     /* Création d'une fenêtre SDL : */
350     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
351         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
352         exit(EXIT_FAILURE);
353     }
354     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
355     ↵ | SDL_DOUBLEBUF)) == NULL) {
356         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
357         ↵ SDL_GetError());
358         SDL_Quit();
359         exit(EXIT_FAILURE);
360     }
361     SDL_WM_SetCaption(titre, NULL);
362     init();
363
364     int active = 1;
365     SDL_Event event;
366     int refresh = 1;
367     int right_grab = 0;
368
369     /* Placement du joueur au centre de la fenêtre : */
370
371     cx = grille_largeur / 2;
372     cy = grille_hauteur / 2;
373     cz = 5;
374
375     SDL_EnableKeyRepeat(50, 50);
```

```
375     while(active) {
376
377         if(refresh) {
378             affichage();
379             SDL_Flip(ecran);
380             refresh = 0;
381         }
382
383         while(SDL_PollEvent(&event)) {
384
385             switch(event.type) {
386                 /* Utilisateur clique sur la croix de la fenêtre : */
387                 case SDL_QUIT : {
388                     active = 0;
389                 } break;
390
391                 /* Utilisateur enfonce une touche du clavier : */
392                 case SDL_KEYDOWN : {
393                     switch(event.key.keysym.sym) {
394                         /* Touche Echap : */
395                         case SDLK_ESCAPE : {
396                             active = 0;
397                         } break;
398
399                         case SDLK_SPACE : {
400                             cx = player_x;
401                             cy = player_y;
402                             refresh = 1;
403                         } break;
404
405                         case SDLK_z :
406                         case SDLK_UP : {
407                             if(can_move(player_x, player_y - 1)) {
408                                 --player_y;
409                                 refresh = 1;
410                                 if(ecran_depuis_camera_y(player_y) < 0) {
411                                     cy -= camera_depuis_ecran_z(hauteur);
412                                 }
413                             }
414                         } break;
415
416                 }
417             }
418
419         }
420
421     }
422 }
```

```

416     case SDLK_s :
417     case SDLK_DOWN : {
418         if(can_move(player_x, player_y + 1)) {
419             ++player_y;
420             refresh = 1;
421             if(ecran_depuis_camera_y(player_y) > hauteur) {
422                 cy += camera_depuis_ecran_z(hauteur);
423             }
424         }
425     } break;

426

427     case SDLK_q :
428     case SDLK_LEFT : {
429         if(can_move(player_x - 1, player_y)) {
430             --player_x;
431             refresh = 1;
432             if(ecran_depuis_camera_x(player_x) < 0) {
433                 cx -= camera_depuis_ecran_z(largeur);
434             }
435         }
436     } break;

437

438     case SDLK_d :
439     case SDLK_RIGHT : {
440         if(can_move(player_x + 1, player_y)) {
441             ++player_x;
442             refresh = 1;
443             if(ecran_depuis_camera_x(player_x) > largeur) {
444                 cx += camera_depuis_ecran_z(largeur);
445             }
446         }
447     } break;
448 }

449 } break;

450

451     case SDL_MOUSEBUTTONDOWN : {
452         switch(event.button.button) {
453             case SDL_BUTTON_WHEELUP : {
454                 cz *= 0.8;
455                 if(cz < 1e-9) {
456                     cz = 1e-9;

```

```
457         }
458         refresh = 1;
459     } break;

460
461     case SDL_BUTTON_WHEELEDOWN : {
462         cz /= 0.8;
463         if(cz > 1e9) {
464             cz = 1e9;
465         }
466         refresh = 1;
467     } break;

468
469     case SDL_BUTTON_RIGHT : {
470         right_grab = 1;
471     } break;
472 }
473 } break;

474
475 case SDL_MOUSEBUTTONDOWN : {
476     switch(event.button.button) {
477         case SDL_BUTTON_RIGHT : {
478             right_grab = 0;
479         } break;
480     }
481 } break;

482
483 case SDL_MOUSEMOTION : {
484     if(right_grab) {
485         cx += camera_depuis_ecran_z(-event.motion.xrel);
486         cy += camera_depuis_ecran_z(-event.motion.yrel);
487         refresh = 1;
488     }
489 } break;
490 }
491 }

492 if(check_finish()) {
493     cx = player_x;
494     cy = player_y;
495     refresh = 1;
496 }
```

```
498     SDL_Delay(1000 / 60);
499 }
500
501     free(*grille);
502     free(grille);
503     grille = NULL;
504
505     SDL_FreeSurface(ecran);
506     SDL_Quit();
507     exit(EXIT_SUCCESS);
508 }
509 }
```

A.9 Consolidation des bases

A.9.1 Entraînement

Correction 83 (★★ Lecture d'arguments en ligne de commande).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int arguments_count, char * arguments_values[]) {
5     int i;
6     for(i = 0; i < arguments_count; ++i) {
7         printf("%d : %s\n", i, arguments_values[i]);
8     }
9     exit(EXIT_SUCCESS);
10 }
```

Correction 84 (★★★ Options depuis ligne de commande).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char * argv[]) {
6     int i;
7     int first = 0;
8     int second = 0;
9     int x = 0, y = 0;
10    for(i = 0; i < argc; ++i) {
11        if(strcmp(argv[i], "--first") == 0 || strcmp(argv[i], "-f") ==
12            → 0) {
13            if(argc - i <= 1) {
14                fprintf(stderr, "Options : Pas assez d'arguments pour
15                    → %s\n", argv[i]);
16                continue;
17            }
18            ++i;
19            first = atoi(argv[i]);
20            continue;
21        }
22    }
23 }
```

```

19 } else if(strcmp(argv[i], "--second") == 0 || strcmp(argv[i],
20   "-s") == 0) {
21   if(argc - i <= 1) {
22     fprintf(stderr, "Options : Pas assez d'arguments pour
23       %s\n", argv[i]);
24     continue;
25   }
26   ++i;
27   second = atoi(argv[i]);
28   continue;
29 } else if(strcmp(argv[i], "--coords") == 0 || strcmp(argv[i],
30   "-c") == 0) {
31   if(argc - i <= 2) {
32     fprintf(stderr, "Options : Pas assez d'arguments pour
33       %s\n", argv[i]);
34     continue;
35   }
36   ++i;
37   x = atoi(argv[i]);
38   ++i;
39   y = atoi(argv[i]);
40   continue;
41 }
printf("first = %d\nsecond = %d\ncoords = (%d, %d)\n", first,
  second, x, y);
exit(EXIT_SUCCESS);
}

```

Correction 85 (★ Allouer dynamiquement une variable).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5   int * variable = NULL;
6   variable = (int *)malloc(sizeof(int));
7   if(variable == NULL) {
8     printf("Erreur allocation : arrêt.\n");
9     exit(EXIT_FAILURE);
}

```

```
10 }
11 *variable = 42;
12 printf("%d\n", *variable);
13 free(variable);
14 variable = NULL;
15 exit(EXIT_SUCCESS);
16 }
```

Correction 86 (★★ Allouer dynamiquement tableau).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     double * tableau = NULL;
6     int taille;
7     printf("Combien d'éléments ? ");
8     scanf("%d", &taille);
9     if((tableau = (double *)calloc(taille, sizeof(double))) == NULL)
10     {
11         printf("Erreur d'allocation de %d éléments : arrêt.\n",
12             → taille);
13         exit(EXIT_FAILURE);
14     }
15     int i;
16     printf("Entrez leurs valeurs : ");
17     for(i = 0; i < taille; ++i) {
18         scanf("%lf", tableau + i);
19     }
20     printf("Bien vos éléments sont notés : [");
21     for(i = 0; i < taille; ++i) {
22         if(i) printf(", ");
23         printf("%g", tableau[i]);
24     }
25     printf("]\n");
26     free(tableau);
27     tableau = NULL;
28     exit(EXIT_SUCCESS);
29 }
```

Correction 87 (★★ Allocation dynamique deux dimensions naïve).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     float ** tableau = NULL;
6     int taille_first = 6, taille_second = 4;
7     int i, j;
8     if((tableau = (float **)malloc(taille_first * sizeof(float *))) == NULL) {
9         printf("Erreur allocation première dimension : arrêt.\n");
10        exit(EXIT_FAILURE);
11    }
12    for(i = 0; i < taille_first; ++i) {
13        if((tableau[i] = (float *)malloc(taille_second * sizeof(float))) == NULL) {
14            printf("Erreur allocation seconde dimension indice %d : "
15                  "arrêt.\n", i);
16            for(--i; i >= 0; --i) {
17                free(tableau[i]);
18            }
19            free(tableau);
20            exit(EXIT_FAILURE);
21        }
22    }
23    for(i = 0; i < taille_first; ++i) {
24        for(j = 0; j < taille_second; ++j) {
25            tableau[i][j] = (float)(i + 1) / (j + 1);
26        }
27    }
28    for(i = 0; i < taille_first; ++i) {
29        for(j = 0; j < taille_second; ++j) {
30            if(j) printf(" ");
31            printf("%.5f", tableau[i][j]);
32        }
33        printf("\n");
34    }
35    for(i = 0; i < taille_first; ++i) {
36        free(tableau[i]);
37    }

```

```
38     free(tableau);
39     tableau = NULL;
40     exit(EXIT_SUCCESS);
41 }
```

Correction 88 (★★ Allocation dynamique deux dimensions en deux allocations).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     float ** tableau = NULL;
6     float * valeurs = NULL;
7     int taille_first = 6, taille_second = 4;
8     int i, j;
9     if((tableau = (float **)malloc(taille_first * sizeof(float *))) == NULL) {
10         printf("Erreur allocation première dimension : arrêt.\n");
11         exit(EXIT_FAILURE);
12     }
13     if((valeurs = (float *)malloc(taille_first * taille_second * sizeof(float))) == NULL) {
14         printf("Erreur allocation seconde dimension : arrêt.\n");
15         free(tableau);
16         exit(EXIT_FAILURE);
17     }
18     for(i = 0; i < taille_first; ++i) {
19         tableau[i] = valeurs + (i * taille_second);
20     }
21     for(i = 0; i < taille_first; ++i) {
22         for(j = 0; j < taille_second; ++j) {
23             tableau[i][j] = (float)(i + 1) / (j + 1);
24         }
25     }
26     for(i = 0; i < taille_first; ++i) {
27         for(j = 0; j < taille_second; ++j) {
28             if(j) printf(" ");
29             printf("%.5f", tableau[i][j]);
30         }
31     }
32     printf("\n");
```

```

32 }
33 free(valeurs);
34 free(tableau);
35 tableau = NULL;
36 exit(EXIT_SUCCESS);
37 }
```

Correction 89 (★★ Allocation dynamique via une fonction).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int alloc_tab(float *** tableau, int taille_first, int
   → taille_second) {
5     float * valeurs = NULL;
6     int i, j;
7     if((*tableau = (float **)malloc(taille_first * sizeof(float *))) == NULL)
   → == NULL) {
8         return 0;
9     }
10    if((valeurs = (float *)malloc(taille_first * taille_second *
   → sizeof(float))) == NULL) {
11        free(*tableau);
12        return 0;
13    }
14    for(i = 0; i < taille_first; ++i) {
15        (*tableau)[i] = valeurs + (i * taille_second);
16    }
17    for(i = 0; i < taille_first; ++i) {
18        for(j = 0; j < taille_second; ++j) {
19            (*tableau)[i][j] = (float)(i + 1) / (j + 1);
20        }
21    }
22    return 1;
23 }
24
25 void free_tab(float *** tableau) {
26     if(tableau == NULL || *tableau == NULL) {
27         return;
28     }
```

```
29     free(**tableau);
30     free(*tableau);
31     *tableau = NULL;
32 }
33
34 int main() {
35     float ** tableau = NULL;
36     int taille_first = 6, taille_second = 4;
37     int i, j;
38     if(! alloc_tab(&tableau, taille_first, taille_second)) {
39         printf("Erreur allocation tableau : arrêt.\n");
40         exit(EXIT_FAILURE);
41     }
42     for(i = 0; i < taille_first; ++i) {
43         for(j = 0; j < taille_second; ++j) {
44             if(j) printf(" ");
45             printf("%.5f", tableau[i][j]);
46         }
47         printf("\n");
48     }
49     free_tab(&tableau);
50     exit(EXIT_SUCCESS);
51 }
```

Correction 90 (★★★ Statistiques sur liste d'entiers).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Déclaration de mini et maxi : variables permettant de */
6     /* connaître les valeurs maximales et minimales du tour de */
7     /* boucle précédent. */
8     int mini = 0, maxi = 0;
9     /* Nous tiendrons à jour le nombre d'entiers positifs lus */
10    /* (pour calcul de moyenne). */
11    int nombre = 0;
12    /* La moyenne non pondérée étant donnée comme la somme des */
13    /* éléments divisée par leur nombre, nous tiendrons à jour */
14    /* la somme des entiers positifs rencontrés. */
15 }
```

```

15  double somme = 0;
16  /* Nous récupérerons l'élément courant fourni par           */
17  /* l'utilisateur.                                              */
18  int current;
19  /* Première saisie : */
20  printf("Entrez des entiers positifs : ");
21  scanf("%d", &current);
22  /* Cette saisie avant la boucle permet de s'assurer d'avoir */
23  /* une valeur valide : un entier positif. Si ce n'est pas le */
24  /* cas, nous n'entrons pas dans la boucle.                      */
25  while(current >= 0) {
26      /* Phase de traitement de l'entier courant fourni (assuré */
27      /* positif par la condition de la boucle.                  */
28      if(nombre == 0) {
29          /* Dans le cas du premier tour de boucle (aucun élément */
30          /* comptabilisé précédemment, nous initialisons mini et */
31          /* maxi avec la valeur de current.                      */
32          mini = current;
33          maxi = current;
34      } else if(current < mini) {
35          /* Sinon si ce n'est pas le premier tour, dans le cas où */
36          /* l'élément est plus petit que la plus petite valeur */
37          /* rencontrée, nous mettons à jour le minimum            */
38          mini = current;
39      } else if(current > maxi) {
40          /* Nous procédons似ilairement pour le maximum que fait */
41          /* pour le minimum                                         */
42          maxi = current;
43      }
44      /* Nous tenons à jour la somme des éléments en ajoutant */
45      /* l'élément courant.                                     */
46      somme += current;
47      /* Nous comptabilisons un élément de plus. */
48      ++nombre;
49      /* Nous lisons à nouveau un entier, s'il est positif ceci */
50      /* relance la boucle et le traitement, sinon ceci arrête */
51      /* la boucle et évite comptabilisation d'un entier       */
52      /* négatif.                                               */
53      scanf("%d", &current);
54  }
55  /* Dans le cas où nous n'effectuons pas une division par */

```

```
56  /* zéro. */  
57  if(nombre) {  
58      /* La somme devient la moyenne */  
59      somme /= nombre;  
60  }  
61  printf("min : %d\n", mini);  
62  printf("max : %d\n", maxi);  
63  printf("moyenne : %g\n", somme);  
64  exit(EXIT_SUCCESS);  
65 }
```

Correction 91 (★★ Extraire de l'information formatée d'une chaîne de caractères).

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main() {  
5     const char * infos = "Linus Torvalds 52 ans C";  
6     /* Nous devons ici extraire les informations d'une chaîne de */  
7     /* caractères non modifiable telle que donnée précédemment */  
8     /* par 'infos'. */  
9     /* Nous préparons les variables dans lesquelles sauvegarder */  
10    /* les informations récupérées. */  
11    char prenom[50]; /* Pour le prénom : mot 1 */  
12    char nom[50]; /* Pour le nom : mot 2 */  
13    int age; /* Pour l'âge : entier 3 */  
14    char langage[50]; /* Pour le langage : mot 5 */  
15    /* sscanf prend en paramètres : */  
16    /* - la chaîne de caractère depuis laquelle lire les */  
17    /* informations. */  
18    /* - la chaîne formatée indiquant le type d'élément à lire. */  
19    /* - les adresses dans lesquelles affecter les résultats. */  
20    /* note : un tableau est l'adresse de son premier élément et */  
21    /* une chaîne de caractère lisible par %s est donc une */  
22    /* adresse */  
23    sscanf(infos, "%s %s %d ans %s", prenom, nom, &age, langage);  
24    printf("Prenom : %s\nNom : %s\nAge : %d\nParle couramment la  
25        ↳ langue %s\n", prenom, nom, age, langage);  
26    exit(EXIT_SUCCESS);  
}
```

Correction 92 (★★★ Mini-interpréteur sur liste d'entiers).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Nous préparons un pointeur qui référencera vers la plage */
6     /* mémoire des données allouée dynamiquement une fois que */
7     /* nous connaîtrons la taille souhaitée. */
8     int * data = NULL;
9     /* Taille des données : donnée par l'utilisateur. */
10    int taille;
11    /* Curseur vers l'élément courant dans nos données. */
12    int i = 0;
13    int j;
14    /* Commande donnée par un caractère. */
15    char op;
16    /* Demande de la taille des données : */
17    printf("taille de la mémoire : ");
18    scanf("%d", &taille);
19    /* Allocation d'une plage mémoire de 'taille' donnée où les */
20    /* valeurs sont initialisées à zéro (ce que fait calloc */
21    /* automatiquement). */
22    /* Notons qu'ici nous affectons data dans la condition par */
23    /* l'adresse renvoyée par calloc. La condition nous permet */
24    /* ici de vérifier si l'allocation a échoué ou non. */
25    /* En cas d'échec c'est NULL qui a été affecté à data et */
26    /* donc (data = ...) vaudrait NULL. */
27    if((data = (int *)calloc(taille, sizeof(int))) == NULL) {
28        /* Dans le cas où l'allocation dynamique échoue, nous */
29        /* l'indiquons à l'utilisateur. */
30        printf("Erreur allocation\n");
31        /* Puis, nous terminons l'exécution. */
32        exit(EXIT_FAILURE);
33    }
34    /* getchar lit un caractère : celui-ci sera utilisé comme */
35    /* commande ou termine si l'utilisateur entre le caractère */
36    /* 'q'. */
37    op = getchar();
38    while(op != 'q') {
39        /* Selon le caractère entré nous lancerons une commande */

```

```
40     /* associée. */  
41     /* Note : les caractères sont des entiers, un caractère */  
42     /* donné entre '' est une constante entière. */  
43     switch(op) {  
44         /* décalage du curseur vers la droite (retour au début) */  
45         /* si dépassement). */  
46         case '>' : i = (i + 1) % taille; break;  
47         /* décalage du curseur vers la gauche (retour en fin si */  
48         /* dépassement). */  
49         case '<' : i = (i + taille - 1) % taille; break;  
50         /* incrémentation de la valeur au curseur. */  
51         case '+' : data[i]++; break;  
52         /* décrémentation de la valeur au curseur. */  
53         case '-' : data[i]--; break;  
54         /* affichage des éléments de la mémoire utilisée. */  
55         case '.' : {  
56             printf("[");  
57             for(j = 0; j < taille; ++j) {  
58                 /* si l'indice j n'est pas 0, ce n'est pas le */  
59                 /* premier tour de boucle. */  
60                 if(j) printf(", ");  
61                 printf("%d", data[j]);  
62             }  
63             printf("]\n");  
64         } break;  
65         /* Toutes les valeurs en mémoire prennent la valeur */  
66         /* regardée par le curseur. */  
67         case '=' : {  
68             for(j = 0; j < taille; ++j) {  
69                 data[j] = data[i];  
70             }  
71         } break;  
72     }  
73     /* On lit la commande pour le tour de boucle suivant. */  
74     op = getchar();  
75 }  
76 /* Toute allocation doit être rendue, on pense à la libérer. */  
77 free(data);  
78 exit(EXIT_SUCCESS);  
79 }
```

Correction 93 (★★★ Jeu du Morpion).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ncurses.h>
4
5 /* clear_grille efface tous les symboles de la grille en les      */
6 /* mettant à ' '                                                 */
7 void clear_grille(char grille[3][3]) {
8     int i;
9     /* Parcours des 9 éléments de la grille. */
10    for(i = 0; i < 9; ++i) {
11        /* Astuce : cet indice peut donner ligne par i / 3 et      */
12        /* colonnes par i % 3.                                         */
13        grille[i / 3][i % 3] = ' ';
14    }
15}
16
17 int finish_grille(char grille[3][3]) {
18     /* v modélisera le vainqueur si trouvé. */
19     char v = ' ';
20     int i;
21     int j;
22     for(i = 0; i < 3; ++i) {
23         /* Parcours sur les lignes : on regarde si sur une même      */
24         /* ligne nous avons qu'un élément et son successeur sont      */
25         /* égaux deux.                                                 */
26         if(grille[i][0] == grille[i][1]
27             && grille[i][1] == grille[i][2]) {
28             /* Si c'est le cas, nous mettons la valeur à laquelle      */
29             /* ils sont tous égaux dans v pour désigner un           */
30             /* vainqueur.                                              */
31             v = grille[i][0];
32             break;
33         }
34         /* Parcours sur les colonnes :似ilairement à celui sur      */
35         /* les lignes.                                               */
36         if(grille[0][i] == grille[1][i]
37             && grille[1][i] == grille[2][i]) {
38             v = grille[0][i];
39             break;

```

```
40     }
41 }
42 /* Vérification diagonales : */
43 if(grille[0][0] == grille[1][1]
44 && grille[1][1] == grille[2][2]) {
45     v = grille[0][0];
46 }
47 if(grille[2][0] == grille[1][1]
48 && grille[1][1] == grille[0][2]) {
49     v = grille[2][0];
50 }
51 /* Selon la valeur de v nous avons un vainqueur ou non. */
52 /* Si le joueur 1 'X' gagne nous revoyons 1. */
53 /* Si le joueur 2 'O' gagne nous revoyons -1. */
54 /* Sinon 0 si pas de vainqueur. */
55 if(v == 'X') {
56     return 1;
57 }
58 if(v == 'O') {
59     return -1;
60 }
61 return 0;
62 }

63
64 void afficher_grille(char grille[3][3]) {
65     int i;
66     int j;
67     /* Itère sur les lignes pour afficher la grille */
68     for(i = 0; i < 4; ++i) {
69         mvprintw(2 * i, 0, "+----+");
70         if(i == 3)
71             break;
72         mvprintw(2 * i + 1, 0, "|%c|%c|%c|", grille[i][0],
73                 grille[i][1], grille[i][2]);
74     }
75 }

76
77 /* placer_grille permet de placer le joueur 1 'X' (valeur 1)    */
78 /* ou le joueur 2 'O' (valeur -1) dans la grille si la coup      */
79 /* est valide.                                                 */
80 int placer_grille(char grille[3][3], int ligne, int colonne,
```

```

81     int joueur) {
82     /* La case n'est pas vide : coup invalide. */
83     if(grille[ligne][colonne] != ' ') {
84         return 0;
85     }
86     /* joueur 1 'X' (valeur 1) place son pion 'X' */
87     if(joueur == 1) {
88         grille[ligne][colonne] = 'X';
89         return 1;
90     }
91     /* joueur 2 'O' (valeur -1) place son pion 'O' */
92     if(joueur == -1) {
93         grille[ligne][colonne] = 'O';
94         return 1;
95     }
96     return 0;
97 }
98
99 int main() {
100    /* grille de taille 3 x 3. */
101    char grille[3][3];
102    /* désignation d'un vainqueur. */
103    int win;
104    /* joueur auquel c'est le tour (1 pour le premier et -1 pour */
105    /* le second). */
106    int joueur = 1;
107    /* position du coup à jouer. */
108    int place;
109    /* nombre de coups déjà joués. */
110    int coups;
111    /* Fonctions d'initialisation d'une fenêtre ncurses. */
112    initscr();
113    noecho();
114    cbreak();
115
116    clear_grille(grille);
117
118    for(coups = 0; coups < 9; ++coups) {
119        /* Affichage de la grille : */
120        clear();
121        afficher_grille(grille);

```

```
122     refresh();
123     /* Récupération d'une caractère via ncurses. */
124     place = getch();
125     /* Une position valide est entre 1 et 9 (pavé numérique). */
126     if(place < '1' || place > '9') {
127         /* Si invalide on relance la boucle : pas de traitement. */
128         --coups;
129         continue;
130     }
131     /* On peut passer d'une représentation {'1' à '9'} à
132     /* {(ligne, colonne)}.
133     /* Pour indices de 0 à 8 (application du - '1' pour les
134     /* obtenir).
135     place -= '1';
136     /* On obtient la ligne par place / 3. (2 - permet de
137     /* reverser le sens en ordonnée)
138     /* 0, 1, 2 donnent 0; 3, 4, 5 donnent 1; 6, 7, 8 donnent 2 */
139     /* Et la colonne par place % 3.
140     /* 0, 3, 6 donnent 0; 1, 4, 7 donnent 1; 2, 5, 8 donnent 2 */
141     if(! placer_grille(grille, 2 - place / 3,
142                         place % 3, joueur)) {
143         /* Si invalide on relance la boucle. */
144         --coups;
145         continue;
146     }
147     /* Coup valide : on change le tour du joueur devant jouer. */
148     joueur *= -1;
149     /* Renvoie un vainqueur (1 ou -1 : valeur de vérité pour
150     /* vrai) sinon 0 (faux)
151     if(win = finish_grille(grille)) {
152         break;
153     }
154 }
155 /* Affiche le gagnant : */
156 clear();
157 afficher_grille(grille);
158 if(win == 0) {
159     myprintw(1, 10, "Pas de gagnant");
160 } else {
161     mvprintw(1, 10, "Le joueur %c gagne !",
162             (win == 1) ? 'X' : 'O');
```

```

163 }
164 refresh();
165 getch();
166
167 /* Termine ncurses proprement : */
168 refresh();
169 clrtoeol();
170 refresh();
171 endwin();
172 exit(EXIT_SUCCESS);
173 }
```

Correction 94 (★★★ Enregistrement et recherche de numéros).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 /* allocStr permet de faire une copie allouée de la chaîne      */
6 /* texte.               */
7 /* À noter qu'une fois que vous savez la coder et que vous      */
8 /* êtes conscients qu'une allocation dynamique y est réalisée,   */
9 /* vous pouvez utiliser strdup pour dupliquer une chaîne de      */
10 /* caractères.          */
11 char * allocStr(const char * texte) {
12     char * res = NULL;
13     /* Allocation de la plage mémoire nécessaire à la sauvegarde */
14     /* des caractères et du marqueur de fin. */
15     if((res = (char *)malloc((1 + strlen(texte)) * sizeof(char)))
16         == NULL) {
17         return NULL;
18     }
19     /* Copie des caractères de la chaîne texte à la chaîne res    */
20     /* (allouée).           */
21     strcpy(res, texte);
22     /* Renvoi de la chaîne allouée. */
23     return res;
24 }
25
26 /* rechercher va regarder dans la table 'noms' de taille      */
```

```
27  /* 'taille' si une chaîne 'texte' existe. */  
28  /* Ceci renvoie un indice positif ou nul si trouvé et -1 */  
29  /* sinon. */  
30  int rechercher(const char * texte, char ** noms, int taille) {  
31      int i;  
32      /* Parcours de éléments de noms */  
33      for(i = 0; i < taille; ++i) {  
34          /* Dans le cas où un élément correspond à la chaîne */  
35          /* recherchée */  
36          if(strcmp(texte, noms[i]) == 0) {  
37              /* nous renvoyons son indice */  
38              return i;  
39          }  
40      }  
41      /* Fin de la boucle : aucun élément de correspond à la */  
42      /* chaîne recherchée, elle n'est pas présente. Nous */  
43      /* renvoyons alors -1. */  
44      return -1;  
45  }  
46  
47  int main() {  
48      /* Tableau dynamique sur des chaînes de caractères. */  
49      char ** noms = NULL;  
50      /* Tableau dynamique sur des entiers. */  
51      long * numeros = NULL;  
52      /* buffer permettant la lecture temporaire d'un nom : chaîne */  
53      /* de caractères. */  
54      char tmpNom[50];  
55      /* variable permettant la lecture temporaire d'un numéro : */  
56      /* entier. */  
57      long tmpNumero;  
58      /* nombre d'éléments et capacité des tableaux dynamiques. */  
59      int taille = 0, capacite = 0;  
60      /* indice dans les tableaux : pour la recherche d'éléments. */  
61      int position;  
62      /* Boucle forever : condition d'arrêt à l'intérieur. */  
63      for(;;) {  
64          /* Lecture d'un nom : */  
65          printf("Nom (None pour arrêter) : ");  
66          scanf("%s", tmpNom);  
67          /* Si l'utilisateur a saisi "None" on arrête la boucle. */
```

```

68     if(strcmp(tmpNom, "None") == 0) {
69         break;
70     }
71     /* Lecture du numéro associé au nom : */
72     printf("Numéro : ");
73     scanf("%ld", &tmpNumero);
74     /* Dans le cas où le nombre d'éléments a atteint la      */
75     /* capacité des tableaux, allons augmenter la capacité      */
76     /* pour accueillir le nouvel élément :                  */
77     if(taille >= capacite) {
78         /* Nous doublons la capacité et ajoutons 10 pour les      */
79         /* petites capacités.                                */
80         capacite = capacite * 2 + 10;
81         /* Réallocation de la liste de noms : */
82         if((noms = (char **)realloc(noms, capacite
83             * sizeof(char *))) == NULL) {
84             printf("Erreur allocation liste des noms\n");
85             exit(EXIT_FAILURE);
86         }
87         /* Réallocation de la liste de numéros : */
88         if((numeros = (long *)realloc(numeros, capacite
89             * sizeof(long))) == NULL) {
90             printf("Erreur allocation liste des numeros\n");
91             exit(EXIT_FAILURE);
92         }
93     }
94     /* Ajout d'une copie du nom lu comme nouvel élément de la */
95     /* liste 'noms'.                                         */
96     if((noms[taille] = allocStr(tmpNom)) == NULL) {
97         printf("Erreur allocation nom \"%s\"\n", tmpNom);
98         exit(EXIT_FAILURE);
99     }
100    /* Ajout du numéro lu à la liste 'numeros'. */
101    numeros[taille] = tmpNumero;
102    /* Comptabilisation d'un élément de plus. */
103    ++taille;
104 }
105 /* Boucle de recherche d'associations nom - numéro : */
106 for(;;) {
107     /* Lecture clavier d'un nom à rechercher : */
108     printf("Nom à rechercher (None pour arrêter) :\n>>> ");

```

```
109     scanf("%s", tmpNom);
110     /* Arrêt si "None" : */
111     if(strcmp(tmpNom, "None") == 0) {
112         break;
113     }
114     /* Récupération de la position dans la liste 'noms'. Si      */
115     /* négatif : non trouvé.                                         */
116     if((position = rechercher(tmpNom, noms, taille)) < 0) {
117         /* On informe l'utilisateur de l'échec de sa requête et */
118         /* on relance la boucle.                                     */
119         printf("\\"%s\\" non trouvé.\n", tmpNom);
120         continue;
121     }
122     /* Nous affichons le numéro associé depuis l'indice      */
123     /* correspondant obtenu dans 'noms'.                         */
124     printf("Le numéro de \"%s\" est %ld\n", tmpNom,
125           numeros[position]);
126 }
127 /* Nous libérons toutes les chaînes allouées : */
128 for(--taille; taille >= 0; --taille) {
129     free(noms[taille]);
130 }
131 /* Nous libérons les tableaux dynamiques : */
132 free(noms);
133 noms = NULL;
134 free(numeros);
135 numeros = NULL;
136 exit(EXIT_SUCCESS);
137 }
```

```
1  /* Alternative avec utilisation de types structurés : */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  /* Association nom - numéro. */
7  typedef struct Node Node;
8  struct Node {
9      char * nom;
10     long numero;
```

```

11 };
```

```

12
13 /* Liste dynamique d'associations nom - numéro. */
14 typedef struct Liste Liste;
15 struct Liste {
16     Node * data;
17     int taille;
18     int capacite;
19 };
20
21 /* Initialisation d'une liste. */
22 Liste Liste_creer();
23
24 /* Ajout d'une association à la liste. */
25 /* Renvoie 0 si échec. */
26 int Liste_ajouter(Liste *, const char * nom, long numero);
27
28 /* Recherche dans la liste par nom. */
29 /* Renvoie 1 et affecte numero si trouvé. */
30 /* Renvoie 0 si échec. */
31 int Liste_rechercher(const Liste *, const char * nom, long *
32     → numero);
33
34 /* Libère la liste. */
35 void Liste_free(Liste *);
```

```

36
37 int main() {
38     Liste liste = Liste_creer();
39     char tmpNom[50];
40     long tmpNumero;
41     for(;;) {
42         printf("Nom (None pour arrêter) : ");
43         scanf("%s", tmpNom);
44         if(strcmp(tmpNom, "None") == 0) {
45             break;
46         }
47         printf("Numéro : ");
48         scanf("%ld", &tmpNumero);
49         if(! Liste_ajouter(&liste, tmpNom, tmpNumero)) {
50             fprintf(stderr, "Erreur ajout de (\\"%s\\", %ld) dans la
51                 → liste.\n", tmpNom, tmpNumero);
52         }
53     }
54 }
```

```
50     Liste_free(&liste);
51     exit(EXIT_FAILURE);
52 }
53 }
54 for(;;) {
55     printf("Nom à rechercher (None pour arrêter) :\n>>> ");
56     scanf("%s", tmpNom);
57     if(strcmp(tmpNom, "None") == 0) {
58         break;
59     }
60     if(! Liste_rechercher(&liste, tmpNom, &tmpNumero)) {
61         printf("\'%s\' non trouvé.\n", tmpNom);
62         continue;
63     }
64     printf("Le numéro de \'%s\' est %ld\n", tmpNom, tmpNumero);
65 }
66 Liste_free(&liste);
67 exit(EXIT_SUCCESS);
68 }

/* Partie implémentation */

69
70 Liste Liste_creer() {
71     Liste res = {NULL, 0, 0};
72     return res;
73 }

74 int Liste_update(Liste * liste) {
75     if(liste->taille >= liste->capacite) {
76         liste->capacite = liste->capacite * 2 + 10;
77         if((liste->data = (Node *)realloc(liste->data, liste->capacite
78             * sizeof(Node))) == NULL) {
79             return 0;
80         }
81     }
82     return 1;
83 }

84 int Liste_ajouter(Liste * liste, const char * nom, long numero) {
85     if(! Liste_update(liste)) {
86         fprintf(stderr, "Erreur extension de la liste.\n");
87     }
88 }
```

```

90     return 0;
91 }
92 if((liste->data[liste->taille].nom = strdup(nom)) == NULL) {
93     fprintf(stderr, "Erreur duplication de \"%s\".\n", nom);
94     return 0;
95 }
96 liste->data[liste->taille].numero = numero;
97 ++(liste->taille);
98 return 1;
99 }

100 int Liste_rechercher(const Liste * liste, const char * nom, long *
101 → numero) {
102     int i;
103     for(i = 0; i < liste->taille; ++i) {
104         if(strcmp(liste->data[i].nom, nom) == 0) {
105             if(numero) {
106                 *numero = liste->data[i].numero;
107             }
108             return 1;
109         }
110     }
111     return 0;
112 }

113 void Liste_free(Liste * liste) {
114     int i;
115     for(i = 0; i < liste->taille; ++i) {
116         free(liste->data[i].nom);
117     }
118     if(liste->data) {
119         free(liste->data);
120         liste->data = NULL;
121     }
122 }
```

Correction 95 (★★★★★ Suite de Fibonacci généralisée et optimisée).

```

1 #include <stdio.h>
2
```

```
3 long n_onacci(int n, int i) {
4     if(i < n) return i == n - 1;
5     long res = 0;
6     int k;
7     for(k = 0; k < n; ++k) {
8         res += n_onacci(n, i - 1 - k);
9     }
10    return res;
11 }
12
13 long n_onacci_better(int n, int i) {
14     long V[n];
15     long tmp;
16     int k;
17     for(k = 0; k < n; ++k) {
18         V[k] = 0;
19     }
20     V[n - 1] = 1;
21     while(i-- > 0) {
22         tmp = 0;
23         for(k = 0; k < n; ++k) {
24             tmp += V[k];
25         }
26         for(k = 0; k < n - 1; ++k) {
27             V[k] = V[k + 1];
28         }
29         V[n - 1] = tmp;
30     }
31     return V[0];
32 }
33
34 void matrix_copy(int n, long dest[n][n], long src[n][n]) {
35     int k, l;
36     for(k = 0; k < n; ++k) {
37         for(l = 0; l < n; ++l) {
38             dest[k][l] = src[k][l];
39         }
40     }
41 }
42
43 void matrix_mul(int n, long res[n][n], long first[n][n], long
44 ← second[n][n]) {
```

```

44 long tmp[n][n];
45 int i, k, l;
46 for(k = 0; k < n; ++k) {
47     for(l = 0; l < n; ++l) {
48         tmp[k][l] = 0;
49         for(i = 0; i < n; ++i) {
50             tmp[k][l] += first[k][i] * second[i][l];
51         }
52     }
53 }
54 matrix_copy(n, res, tmp);
55 }

56 void matrix_identity(int n, long res[n][n]) {
57     int k, l;
58
59     for(k = 0; k < n; ++k) {
60         for(l = 0; l < n; ++l) {
61             res[k][l] = 0;
62         }
63     }
64
65     for(k = 0; k < n; ++k) {
66         res[k][k] = 1;
67     }
68 }

69 long n_onacci_log(int n, int i) {
70     long M[n][n];
71     long res[n][n];
72     int k, l;
73
74     for(k = 0; k < n; ++k) {
75         M[0][k] = 1;
76         for(l = 1; l < n; ++l) {
77             M[l][k] = 0;
78         }
79     }
80
81     for(k = 1; k < n; ++k) {
82         M[k][k - 1] = 1;
83     }
84

```

```
85     matrix_identity(n, res);
86
87     while(i > 0) {
88         if(i % 2 == 1) {
89             matrix_mul(n, res, res, M);
90         }
91         matrix_mul(n, M, M, M);
92         i /= 2;
93     }
94
95     return res[n - 1][0];
96 }
97
98 int main() {
99     int degree;
100    printf("Degré de la suite : ");
101    scanf("%d", &degree);
102    int i;
103    printf("[");
104    for(i = 0; i < 20; ++i) {
105        if(i) printf(", ");
106        printf("%ld", n_onacci(degree, i));
107    }
108    printf("]\n");
109    printf("[");
110    for(i = 0; i < 20; ++i) {
111        if(i) printf(", ");
112        printf("%ld", n_onacci_better(degree, i));
113    }
114    printf("]\n");
115    printf("[");
116    for(i = 0; i < 20; ++i) {
117        if(i) printf(", ");
118        printf("%ld", n_onacci_log(degree, i));
119    }
120    printf("]\n");
121    return 0;
122 }
```

A.10 Fichiers

A.10.1 Entraînement

Correction 96 (★ Création d'un fichier).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     const char * chemin = "test.txt";
6     FILE * fichier = NULL;
7     if((fichier = fopen(chemin, "w")) == NULL) {
8         fprintf(stderr, "Erreur creation \"%s\" : arrêt.\n", chemin);
9         exit(EXIT_SUCCESS);
10    }
11    fprintf(fichier, "Test.\n");
12    fclose(fichier);
13    fichier = NULL;
14    exit(EXIT_SUCCESS);
15 }
```

Correction 97 (★★ Lire des fichiers).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char * argv[]) {
5     FILE * output = stdout;
6     FILE * current = NULL;
7     int i;
8     int car;
9     for(i = 1; i < argc; ++i) {
10        if((current = fopen(argv[i], "r")) == NULL) {
11            fprintf(stderr, "Erreur ouverture \"%s\".\n", argv[i]);
12            continue;
13        }
14        while((car = fgetc(current)) != EOF) {
15            fputc(car, output);
```

```
16     }
17     fclose(current);
18 }
19 exit(EXIT_SUCCESS);
20 }
```

Correction 98 (★★ Fichier en binaire).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     const char * path = "nombre.bin";
6     FILE * fichier = NULL;
7     int compteur = 0;
8     if((fichier = fopen(path, "rb")) != NULL) {
9         if(fread(&compteur, sizeof(int), 1, fichier) != 1) {
10            fprintf(stderr, "Erreur lecture dans %s\n", path);
11        }
12        fclose(fichier);
13    }
14    ++compteur;
15    if((fichier = fopen(path, "wb")) == NULL) {
16        fprintf(stderr, "Erreur ouverture %s\n", path);
17        exit(EXIT_FAILURE);
18    }
19    if(fwrite(&compteur, sizeof(int), 1, fichier) != 1) {
20        fprintf(stderr, "Erreur écriture dans %s\n", path);
21        fclose(fichier);
22        exit(EXIT_FAILURE);
23    }
24    fclose(fichier);
25    fichier = NULL;
26    printf("Programme lancé %d fois\n", compteur);
27    exit(EXIT_SUCCESS);
28 }
```

Correction 99 (★★ Éditer un fichier).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     const char * path = "exemple.txt";
6     FILE * fichier = NULL;
7     int car;
8     if((fichier = fopen(path, "r+")) == NULL) {
9         fprintf(stderr, "Erreur ouverture %s\n", path);
10        exit(EXIT_FAILURE);
11    }
12    while((car = fgetc(fichier)) != EOF) {
13        if(car >= 'a' && car <= 'z') {
14            fseek(fichier, -1, SEEK_CUR);
15            fputc(car - 'a' + 'A', fichier);
16        }
17    }
18    fclose(fichier);
19    fichier = NULL;
20    exit(EXIT_SUCCESS);
21 }
```

Correction 100 (★ Compteur de lancements).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     /* Chemin du fichier où sauvegarder le nombre d'ouvertures. */
6     const char * sauvegarde = "compteur.txt";
7     /* Référence vers un fichier. */
8     FILE * fich = NULL;
9     /* Par défaut le nombre d'ouverture est 0. */
10    int count = 0;
11    /* Dans le cas où nous arrivons à ouvrir le fichier en      */
12    /* lecture.                                              */
13    if((fich = fopen(sauvegarde, "r")) != NULL) {
14        /* Nous récupérons la valeur inscrite pour mettre à jour */
15        /* le compteur d'ouvertures.                           */
16        fscanf(fich, "%d", &count);
```

```
17     /* Et nous fermons le fichier. */
18     fclose(fich);
19     fich = NULL;
20 }
21 /* Nous avons le nombre de fois où le fichier a été ouvert      */
22 /* précédemment.                                                 */
23 /* Nous ouvrons cette fois le fichier en écriture pour          */
24 /* mettre à jour son contenu.                                     */
25 if((fich = fopen(sauvegarde, "w+")) == NULL) {
26     fprintf(stderr, "Erreur ouverture \"%s\"\n", sauvegarde);
27     exit(EXIT_FAILURE);
28 }
29 /* Nous comptons un lancement de plus. */
30 ++count;
31 printf("Programme lancé %d fois\n", count);
32 /* Nous inscrivons la valeur à jour dans le fichier. */
33 fprintf(fich, "%d\n", count);
34 fclose(fich);
35 fich = NULL;
36 exit(EXIT_SUCCESS);
37 }
```

Correction 101 (★★★ Codage Vigenère depuis fichier).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void vigenere(FILE * input, char * cle, int sign, FILE * output) {
6     int i, j;
7     int value;
8     int offset;
9     int texte;
10    int message;
11    /* Nous lisons des caractères dans le fichier tant que nous   */
12    /* ne rencontrons pas le marqueur de fin de fichier.           */
13    for(i = 0, j = 0; (texte = fgetc(input)) != EOF; ++i) {
14        /* Si le caractère est une majuscule */
15        if(texte >= 'A' && texte <= 'Z') {
16            /* Nous récupérons son décalage par rapport à 'A' de sorte */
```

```

17  /* que 'A' donne 0, 'B' donne 1, et ainsi de suite. */
18  value = texte - 'A';
19  /* Nous mettons le résultat à 'A' pour appliquer le
20  /* décalage ensuite et conserver la connaissance d'une
21  /* majuscule. */
22  message = 'A';
23 } else if(texte >= 'a' && texte <= 'z') {
24  /* Nous procérons similairement aux majuscules dans le cas */
25  /* d'une lettre minuscule. */
26  value = texte - 'a';
27  message = 'a';
28 } else {
29  /* En cas d'un caractère non alphabétique, nous
30  /* l'imprimons sans appliquer de décalage.
31  fputc(texte, output);
32  continue;
33 }
34 /* Nous récupérons le décalage correspondant à la clé. */
35 if(cle[j] >= 'A' && cle[j] <= 'Z') {
36  offset = cle[j] - 'A';
37 } else if(cle[j] >= 'a' && cle[j] <= 'z') {
38  offset = cle[j] - 'a';
39 }
40 /* Nous calculons le décalage depuis la clé. */
41 value = ((value + sign * offset) % 26 + 26) % 26;
42 /* Nous ajoutons le décalage à la lettre. */
43 message += value;
44 /* Nous mettons à jour le curseur sur les lettres de la clé. */
45 j = (j + 1) % strlen(cle);
46 /* Nous imprimons le résultat. */
47 fputc(message, output);
48 }
49 }

50 int main(int argc, char * argv[]) {
51 if(argc <= 2) {
52  printf("Attendu : %s [FICHIER MESSAGE] [CLE]\n", argv[0]);
53  printf("Attendu : %s [FICHIER MESSAGE] [CLE] decode\n",
54  → argv[0]);
55  exit(EXIT_FAILURE);
56 }
```

```
57  /* Nous ajoutons de la sémantique aux arguments obtenus. */
58  char * path = argv[1];
59  char * cle = argv[2];
60  /* Nous déterminons si le décalage doit s'appliquer vers      */
61  /* l'avant ou vers l'arrière.                                     */
62  int sign = (argc > 3 && strcmp(argv[3], "decode") == 0) ? -1 :
63  →   1;
64  FILE * input = NULL;
65  FILE * output = stdout;
66  /* Nous ouvrons le fichier donné en entrée. */
67  if((input = fopen(path, "r")) == NULL) {
68      fprintf(stderr, "Erreur ouverture \"%s\"\n", path);
69      exit(EXIT_FAILURE);
70  }
71  /* Nous appliquons le codage vigénère depuis la récupération */
72  /* dans le fichier.                                         */
73  vigenere(input, cle, sign, output);
74  fclose(input);
75  exit(EXIT_SUCCESS);
```

Correction 102 (★★★ Sauvegarde et chargement en binaire).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int savePerso(FILE * output, const char * nom, int vie, int atk,
→   int def, int vit) {
6     int tailleNom = strlen(nom);
7     /* Nous sauvegardons les éléments constituants le personnage */
8     /* en écriture binaire.                                         */
9     /* fwrite prend une adresse sur les données à écrire, la      */
10    /* taille d'un élément, le nombre d'éléments et où l'écrire. */
11    /* Nous vérifions que le nombre d'éléments souhaités ont      */
12    /* bien été écrits.                                         */
13    if(fwrite(&tailleNom, sizeof(int), 1, output) != 1
14    || fwrite(nom, sizeof(char), tailleNom, output) != tailleNom
15    || fwrite(&vie, sizeof(int), 1, output) != 1
16    || fwrite(&atk, sizeof(int), 1, output) != 1
```

```

17    || fwrite(&def, sizeof(int), 1, output) != 1
18    || fwrite(&vit, sizeof(int), 1, output) != 1) {
19        fprintf(stderr, "Erreur d'écriture du personnage \"%s\".\n",
20                nom);
21        return 0;
22    }
23    return 1;
24}

25 int loadPerso(FILE * input, char ** nom, int * vie, int * atk, int
26    * def, int * vit) {
27    int tailleNom;
28    /* fread fonctionne似然性allyer à fwrite. */
29    /* Nous récupérons la taille. */
30    if(fread(&tailleNom, sizeof(int), 1, input) != 1) {
31        fprintf(stderr, "Erreur de lecture d'un personnage.\n");
32        return 0;
33    }
34    /* Depuis la connaissance de la taille, nous allons la plage */
35    /* mémoire pour recevoir le nom inscrit dans le fichier. */
36    if((*nom = (char *)malloc((tailleNom + 1) * sizeof(char))) ==
37        NULL) {
38        fprintf(stderr, "Erreur alloc nom\n");
39        return 0;
40    }
41    /* Nous lisons le nom depuis le fichier : affectation des */
42    /* caractères. */
43    if(fread(*nom, sizeof(char), tailleNom, input) != tailleNom) {
44        fprintf(stderr, "Erreur de lecture d'un personnage.\n");
45        free(*nom);
46        *nom = NULL;
47        return 0;
48    }
49    (*nom)[tailleNom] = '\0';
50    /* Nous lisons les autres stats. */
51    if(fread(vie, sizeof(int), 1, input) != 1
52    || fread(atk, sizeof(int), 1, input) != 1
53    || fread(def, sizeof(int), 1, input) != 1
      || fread(vit, sizeof(int), 1, input) != 1) {
        fprintf(stderr, "Erreur de lecture du personnage \"%s\".\n",
                *nom);
    }
}

```

```
54     free(*nom);
55     *nom = NULL;
56     return 0;
57 }
58 return 1;
59 }

60
61 int main(int argc, char * argv[]) {
62     if(argc <= 1) {
63         printf("Attendu :\n"
64             "\t%-create [FICHIER] [NOM] [VIE] [ATK] [DEF] [VIT]\n"
65             "\t%-read [FICHIER]\n", argv[0], argv[0]);
66         exit(EXIT_FAILURE);
67     }
68     if(strcmp(argv[1], "-create") == 0 && argc > 7) {
69         FILE * output = NULL;
70         if((output = fopen(argv[2], "w+")) == NULL) {
71             fprintf(stderr, "Erreur ouverture \"%s\"\n", argv[2]);
72             exit(EXIT_FAILURE);
73         }
74         savePerso(output, argv[3], atoi(argv[4]), atoi(argv[5]),
75             atoi(argv[6]), atoi(argv[7]));
76         fclose(output);
77     } else if(strcmp(argv[1], "-read") == 0 && argc > 2) {
78         char * nom = NULL;
79         int vie, atk, def, vit;
80         FILE * input = NULL;
81         if((input = fopen(argv[2], "r")) == NULL) {
82             fprintf(stderr, "Erreur ouverture \"%s\"\n", argv[2]);
83             exit(EXIT_FAILURE);
84         }
85         loadPerso(input, &nom, &vie, &atk, &def, &vit);
86         printf("Personnage : {\n"
87             "    Nom : %s\n"
88             "    Vie : %d\n"
89             "    Attaque : %d\n"
90             "    Défense : %d\n"
91             "    Vitesse : %d\n"
92             "}\n", nom, vie, atk, def, vit);
93         free(nom);
94         nom = NULL;
```

```

94     fclose(input);
95 }
96 exit(EXIT_SUCCESS);
97 }
```

Correction 103 (★★★ Enregistrement et recherche de numéros avec sauvegarde).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 char * allocStr(const char * texte) {
6     char * res = NULL;
7     if((res = (char *)malloc((1 + strlen(texte)) * sizeof(char))) ==
8         → NULL) {
9         return NULL;
10    }
11    strcpy(res, texte);
12    return res;
13 }
14
15 int rechercher(const char * texte, char ** noms, int taille) {
16     int i;
17     for(i = 0; i < taille; ++i) {
18         if(strcmp(texte, noms[i]) == 0) {
19             return i;
20         }
21     }
22     return -1;
23 }
24
25 /* L'ajout pouvant se faire depuis la lecture d'un fichier et */
26 /* la saisie utilisateur, nous factorisons le code pour           */
27 /* réaliser l'ajout par cette fonction.                         */
28 void ajouter(const char * nom, long numero, char *** noms, long **
29   → numeros, int * taille, int * capacite) {
30     if(*taille >= *capacite) {
31         *capacite = *capacite * 2 + 10;
32         if((*noms = (char **)realloc(*noms, *capacite * sizeof(char
33           → *)))) == NULL) {
```

```
31     printf("Erreur allocation liste des noms\n");
32     exit(EXIT_FAILURE);
33 }
34 if((*numeros = (long *)realloc(*numeros, *capacite *
35     sizeof(long))) == NULL) {
36     printf("Erreur allocation liste des numeros\n");
37     exit(EXIT_FAILURE);
38 }
39 if(((*noms)[*taille] = allocStr(nom)) == NULL) {
40     printf("Erreur allocation nom \"%s\"\n", nom);
41     exit(EXIT_FAILURE);
42 }
43 (*numeros)[*taille] = numero;
44 ++(*taille);
45 }

46 int main(int argc, char * argv[]) {
47     char ** noms = NULL;
48     long * numeros = NULL;
49     char tmpNom[50];
50     long tmpNumero;
51     int taille = 0, capacite = 0;
52     int position;
53     char * inputPath = NULL;
54     char * outputPath = NULL;
55     FILE * input = NULL;
56     FILE * output = NULL;
57     int i;
58     /* Nous bouclons sur les arguments à la recherche des */
59     /* options possibles */
60     for(i = 1; i < argc - 1; ++i) {
61         if(strcmp(argv[i], "-i") == 0) {
62             inputPath = argv[i + 1];
63             ++i;
64             continue;
65         } else if(strcmp(argv[i], "-o") == 0) {
66             outputPath = argv[i + 1];
67             ++i;
68             continue;
69         }
70 }
```

```

1 } }
2 /* Si fichier d'entrée est fourni, nous ajoutons les
3 /* éléments depuis ce fichier.
4 if(inputPath) {
5     if((input = fopen(inputPath, "r")) != NULL) {
6         fscanf(input, "%s", tmpNom);
7         while(strcmp(tmpNom, "None") != 0) {
8             fscanf(input, "%ld", &tmpNumero);
9             ajouter(tmpNom, tmpNumero, &noms, &numeros, &taille,
10            → &capacite);
11            fscanf(input, "%s", tmpNom);
12        }
13        fclose(input);
14        input = NULL;
15    } else {
16        fprintf(stderr, "Ouverture en lecture de \"%s\""
17           → impossible.\n", outputPath);
18    }
19 }
20 /* Nous ajoutons les éléments saisis par l'utilisateur. */
21 for(;;) {
22     printf("Nom (None pour arrêter) : ");
23     scanf("%s", tmpNom);
24     if(strcmp(tmpNom, "None") == 0) {
25         break;
26     }
27     printf("Numéro : ");
28     scanf("%ld", &tmpNumero);
29     ajouter(tmpNom, tmpNumero, &noms, &numeros, &taille,
30            → &capacite);
31 }
32 /* Nous sauvegardons les ajouts de l'utilisateur. */
33 if(outputPath) {
34     if((output = fopen(outputPath, "w+")) != NULL) {
35         for(i = 0; i < taille; ++i) {
36             fprintf(output, "%s %ld\n", noms[i], numeros[i]);
37         }
38         fprintf(output, "None\n");
39         fclose(output);
40         output = NULL;
41     } else {
42

```

```
109     fprintf(stderr, "Ouverture en écriture de \"%s\"  
110     ↪ impossible.\n", outputPath);  
111 }  
112 /* Nous laissons l'utilisateur rechercher des associations */  
113 /* depuis des noms. */  
114 for(;;) {  
115     printf("Nom à rechercher (None pour arrêter) :\n>>> ");  
116     scanf("%s", tmpNom);  
117     if(strcmp(tmpNom, "None") == 0) {  
118         break;  
119     }  
120     if((position = rechercher(tmpNom, noms, taille)) < 0) {  
121         printf("\'%s\' non trouvé.\n", tmpNom);  
122         continue;  
123     }  
124     printf("Le numéro de \"%s\" est %ld\n", tmpNom,  
125     ↪ numeros[position]);  
126 }  
127 for(--taille; taille >= 0; --taille) {  
128     free(noms[taille]);  
129 }  
130 free(noms);  
131 noms = NULL;  
132 free(numeros);  
133 numeros = NULL;  
134 exit(EXIT_SUCCESS);  
135 }
```

A.11 Structures

A.11.1 Entraînement

Correction 104 (★ Créer une structure).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Stats Stats;
5 struct Stats {
6     int vie;
7     int attaque;
8     int defense;
9     int vitesse;
10 };
11
12 int main() {
13     Stats perso;
14     perso.vie = 100;
15     perso.attaque = 50;
16     perso.defense = 70;
17     perso.vitesse = 30;
18     printf("Perso : {\n");
19     printf("\tvie : %d\n", perso.vie);
20     printf("\tattaque : %d\n", perso.attaque);
21     printf("\tdefense : %d\n", perso.defense);
22     printf("\tvitesse : %d\n", perso.vitesse);
23     printf("}\n");
24     exit(EXIT_SUCCESS);
25 }
```

Correction 105 (★★ Passage par copie).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Stats Stats;
5 struct Stats {
```

```
6     int vie;
7     int attaque;
8     int defense;
9     int vitesse;
10    };
11
12 Stats Stats_creer(int vie, int attaque, int defense, int vitesse)
13 {  
14     Stats perso;  
15     perso.vie = vie;  
16     perso.attaque = attaque;  
17     perso.defense = defense;  
18     perso.vitesse = vitesse;  
19     return perso;  
20 }
21
22 void Stats_afficher(const Stats perso) {  
23     printf("Perso : {\n");  
24     printf("\tvie : %d\n", perso.vie);  
25     printf("\tattaque : %d\n", perso.attaque);  
26     printf("\tdefense : %d\n", perso.defense);  
27     printf("\tvitesse : %d\n", perso.vitesse);  
28     printf("}\n");  
29 }
30
31 int main() {  
32     Stats perso = Stats_creer(100, 50, 70, 30);  
33     Stats_afficher(perso);  
34     exit(EXIT_SUCCESS);  
}
```

Correction 106 (★ Passage par adresse).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Stats Stats;
5 struct Stats {
6     int vie;
7     int attaque;
```

```

8   int defense;
9   int vitesse;
10 }
11
12 int Stats_creer(Stats * perso, int vie, int attaque, int defense,
13   int vitesse) {
14   if(vie < 0 || attaque < 0 || defense < 0 || vitesse < 0) {
15     fprintf(stderr, "Les statistiques ne peuvent pas être
16     négatives.\n");
17     return 0;
18   }
19   perso->vie = vie;
20   perso->attaque = attaque;
21   perso->defense = defense;
22   perso->vitesse = vitesse;
23   return 1;
24 }
25
26 void Stats_afficher(const Stats * perso) {
27   printf("Perso : {\n");
28   printf("\tvie : %d\n", perso->vie);
29   printf("\tattaque : %d\n", perso->attaque);
30   printf("\tdefense : %d\n", perso->defense);
31   printf("\tvitesse : %d\n", perso->vitesse);
32   printf("}\n");
33 }
34
35 int main() {
36   Stats perso;
37   if(! Stats_creer(&perso, 100, 50, 70, 30)) {
38     fprintf(stderr, "Erreur création personnage : arrêt.\n");
39     exit(EXIT_FAILURE);
40   }
41   Stats_afficher(&perso);
42   exit(EXIT_SUCCESS);
43 }
```

Correction 107 (★★ Manipulation dynamique).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Stats Stats;
5 struct Stats {
6     int vie;
7     int attaque;
8     int defense;
9     int vitesse;
10 };
11
12 Stats * Stats_creer(int vie, int attaque, int defense, int
13 → vitesse) {
14     Stats * perso = NULL;
15     if(vie < 0 || attaque < 0 || defense < 0 || vitesse < 0) {
16         fprintf(stderr, "Les statistiques ne peuvent pas être
17 → négatives.\n");
18         return NULL;
19     }
20     if((perso = (Stats *)malloc(sizeof(Stats))) == NULL) {
21         fprintf(stderr, "Erreur allocation Stats.\n");
22         return NULL;
23     }
24     perso->vie = vie;
25     perso->attaque = attaque;
26     perso->defense = defense;
27     perso->vitesse = vitesse;
28     return perso;
29 }
30
31 void Stats_free(Stats ** perso) {
32     if(perso == NULL || *perso == NULL) {
33         return;
34     }
35     free(*perso);
36     *perso = NULL;
37 }
38
39 void Stats_afficher(const Stats * perso) {
40     printf("Perso : {\n");
41     printf("\tvie : %d\n", perso->vie);
```

```

40 printf("\tattaque : %d\n", perso->attaque);
41 printf("\tdefense : %d\n", perso->defense);
42 printf("\tvitesse : %d\n", perso->vitesse);
43 printf("}\n");
44 }
45
46 int main() {
47     Stats * perso = NULL;
48     if((perso = Stats_creer(100, 50, 70, 30)) == NULL) {
49         fprintf(stderr, "Erreur création personnage : arrêt.\n");
50         exit(EXIT_FAILURE);
51     }
52     Stats_afficher(perso);
53     Stats_free(&perso);
54     exit(EXIT_SUCCESS);
55 }
```

Correction 108 (★★★ Définir une structure Vecteur2d).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define M_PI 3.14159265359
5 #include <math.h>
6
7 /* Nous utilisons un typedef pour s'éviter l'écriture struct */
8 /* Vecteur2d dans la suite et la remplacer par une écriture */
9 /* Vecteur2d. */
10 typedef struct Vecteur2d Vecteur2d;
11 /* Nous fabriquons une structure avec deux champs : */
12 struct Vecteur2d {
13     double x; /* un champ x réel */
14     double y; /* un champ y réel */
15 };
16
17 /* Fonction d'initialisation d'un Vecteur2d depuis des */
18 /* coordonnées x et y. */
19 Vecteur2d Vecteur2d_create(double x, double y) {
20     /* Cette syntaxe peut être utilisée à la définition d'une */
21     /* structure comme pour un tableau. */
22 }
```

```
22 Vecteur2d p = {x, y};  
23 /* Nous renvoyons une copie de la structure créée. */  
24 return p;  
25 }  
26  
27 /* Compatible avec Vecteur2d_create.  
28 /* Les structures sont ici passées par copie et donc ceci ne  
29 /* modifiera pas la structure envoyée.  
30 Vecteur2d Vecteur2d_translation(Vecteur2d target, Vecteur2d  
→ translation) {  
31     target.x += translation.x;  
32     target.y += translation.y;  
33     /* d'où le besoin de la renvoyer pour affectation : */  
34     return target;  
35 }  
36  
37 Vecteur2d Vecteur2d_scale(Vecteur2d target, Vecteur2d origin,  
→ double scale) {  
38     target.x = scale * (target.x - origin.x) + origin.x;  
39     target.y = scale * (target.y - origin.y) + origin.y;  
40     return target;  
41 }  
42  
43 Vecteur2d Vecteur2d_rotation(Vecteur2d target, Vecteur2d origin,  
→ double angleDegree) {  
44     Vecteur2d res;  
45     double angle = (angleDegree * M_PI) / 180;  
46     res.x = cos(angle) * (target.x - origin.x) - sin(angle) *  
→ (target.y - origin.y) + origin.x;  
47     res.y = sin(angle) * (target.x - origin.x) + cos(angle) *  
→ (target.y - origin.y) + origin.y;  
48     return res;  
49 }  
50  
51 void Vecteur2d_print(Vecteur2d vec) {  
52     printf("(%.g, %.g)\n", vec.x, vec.y);  
53 }  
54  
55 int main() {  
56     Vecteur2d p = Vecteur2d_create(0, 0);  
57     printf("Vecteur2d : ");
```

```

58 Vecteur2d_print(p);
59 p = Vecteur2d_translation(p, Vecteur2d_create(1, 2));
60 printf("Translation par un Vecteur2d : ");
61 Vecteur2d_print(Vecteur2d_create(1, 2));
62 Vecteur2d_print(p);
63 p = Vecteur2d_scale(p, Vecteur2d_create(1, 0), 0.5);
64 printf("Agrandissement de rapport %g et de centre un Vecteur2d :
   → ", 0.5);
65 Vecteur2d_print(Vecteur2d_create(1, 0));
66 Vecteur2d_print(p);
67 p = Vecteur2d_rotation(p, Vecteur2d_create(0, 2), 135);
68 printf("Rotation d'angle %d deg et de centre un Vecteur2d : ",
   → 135);
69 Vecteur2d_print(Vecteur2d_create(0, 2));
70 Vecteur2d_print(p);
71 exit(EXIT_SUCCESS);
72 }
```

Correction 109 (★★★ Structure pour gérer une grille).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ncurses.h>
4
5 typedef struct Grille Grille;
6 struct Grille {
7     /* Notre grille sera modélisée en interne par un tableau à      */
8     /* une dimension :                                              */
9     char * grille;
10    int largeur;
11    int hauteur;
12};
13
14 /* donne un pointeur sur une case de la grille */
15 char * Grille_case(const Grille * grille, int x, int y);
16
17 /* crée une grille de taille donnée */
18 Grille Grille_creer(int largeur, int hauteur);
19
20 /* affiche une grille à l'écran */
```

```
21 void Grille_afficher(const Grille * grille);
22
23 /* libère une grille */
24 void Grille_free(Grille * grille);
25
26 int main() {
27     int largeur = 60, hauteur = 20;
28     Grille grille = Grille_creer(largeur, hauteur);
29     int x = 1, y = 1;
30     initscr();
31     noecho();
32     cbreak();
33     do {
34         clear();
35         Grille_afficher(&grille);
36         mvprintw(y, x, "@");
37         mvprintw(y, x, " ");
38         refresh();
39         getch();
40         /* gestion des événements */
41     } while(1);
42     refresh();
43     clrtoeol();
44     refresh();
45     endwin();
46     Grille_free(&grille);
47     exit(EXIT_SUCCESS);
48 }
49
50 /* Pour modifier la grille, nous proposons d'obtenir un      */
51 /* pointeur vers un élément de la grille depuis les      */
52 /* coordonnées (x, y) de la case de celle-ci. Ceci nous permet */
53 /* de rester indépendant de la représentation de la grille en */
54 /* mémoire (1D, 2D, ...)                                     */
55 char * Grille_case(const Grille * grille, int x, int y) {
56     if(grille == NULL || x < 0 || y < 0 || x >= grille->largeur || y
57     >= grille->hauteur) {
58         return NULL;
59     }
60     return grille->grille + (x + y * grille->largeur);
61 }
```

```

61
62 Grille Grille_creer(int largeur, int hauteur) {
63     Grille grille = {NULL, 0, 0};
64     if((grille.grille = (char *)malloc(sizeof(char) * largeur *
65         → hauteur)) == NULL) {
66         printf("Allocation de la grille impossible.\n");
67         exit(EXIT_FAILURE);
68     }
69     grille.largeur = largeur;
70     grille.hauteur = hauteur;
71     int x, y;
72     for(x = 0; x < largeur * hauteur; ++x) {
73         grille.grille[x] = ' ';
74     }
75     for(y = 0; y < hauteur; ++y) {
76         *Grille_case(&grille, 0, y) = '#';
77         *Grille_case(&grille, largeur - 1, y) = '#';
78     }
79     for(x = 1; x < largeur - 1; ++x) {
80         *Grille_case(&grille, x, 0) = '#';
81         *Grille_case(&grille, x, hauteur - 1) = '#';
82     }
83     return grille;
84 }
85 void Grille_afficher(const Grille * grille) {
86     int x, y;
87     for(y = 0; y < grille->hauteur; ++y) {
88         for(x = 0; x < grille->largeur; ++x) {
89             mvprintw(y, x, "%c", *Grille_case(grille, x, y));
90         }
91     }
92 }
93
94 void Grille_free(Grille * grille) {
95     if(grille == NULL) {
96         return;
97     }
98     free(grille->grille);
99     grille->grille = NULL;
100    grille->largeur = 0;

```

```
101     grille->hauteur = 0;  
102 }
```

Correction 110 (★★★ Gérer un combat de personnages).

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3 #include <string.h>  
4 #include <time.h>  
5  
6 int min(int a, int b) {  
7     return (a < b) ? a : b;  
8 }  
9  
10 int max(int a, int b) {  
11     return (a > b) ? a : b;  
12 }  
13  
14 /* Nous définissons une structure représentant les stats d'un */  
15 /* personnage à un instant donné */  
16 typedef struct Stats Stats;  
17 struct Stats {  
18     int vie;  
19     int atk;  
20     int def;  
21     int vit;  
22 };  
23  
24 /* Nous énumérons les différents stats définis plus haut. */  
25 typedef enum {  
26     STI_VIE,  
27     STI_ATK,  
28     STI_DEF,  
29     STI_VIT,  
30     STI_NONE  
31 } StatsId;  
32  
33 Stats Stats_creer(int vie, int atk, int def, int vit) {  
34     Stats stats = {vie, atk, def, vit};  
35     return stats;
```

```

36 }
37
38 /* Nous utilisons les stats précédemment donnés pour           */
39 /* sauvegarder des stats au début d'un combat et l'état          */
40 /* actuelle de ceux-ci. */                                     */
41 typedef struct Personnage Personnage;
42 struct Personnage {
43     char nom[20];
44     Stats base;
45     Stats current;
46 };
47
48 /* Nous énumérons les sorts disponibles. */
49 typedef enum {
50     SOI_COUP = 1,
51     SOI_LOW_ATK,
52     SOI_LOW_DEF,
53     SOI_LOW_VIT,
54     SOI_SOIN,
55     SOI_UP_ATK,
56     SOI_UP_DEF,
57     SOI_UP_VIT
58 } SortId;
59
60 Personnage Personnage_creer(const char * nom, Stats base) {
61     Personnage perso = {"", base, base};
62     strcpy(perso.nom, nom);
63     return perso;
64 }
65
66 void Personnage_fullheal(Personnage * perso) {
67     perso->current = perso->base;
68 }
69
70 void Personnage_stat_mul(Personnage * perso, StatsId stat, double
71     → coeff) {
72     switch(stat) {
73         case STI_VIE : {
74             perso->current.vie = min(perso->current.vie * coeff,
75                 → perso->base.vie);
76         } break;
77     }
78 }
```



```

106     printf("-----+\n");
107 }
108
109 void Personnages_appliquer_sort(SortId id, Personnage * tori,
110 → Personnage * uke) {
111     if(! Personnage_vivant(tori))
112         return;
113     switch(id) {
114         case SOI_COUP : {
115             Personnage_attacks_other(tori, uke);
116             printf("%s attaque %s.\n", tori->nom, uke->nom);
117         } break;
118         case SOI_LOW_ATK : {
119             Personnage_stat_mul(uke, STI_ATK, 0.8);
120             printf("L'attaque de %s diminue.\n", uke->nom);
121         } break;
122         case SOI_LOW_DEF : {
123             Personnage_stat_mul(uke, STI_DEF, 0.6);
124             printf("La defense de %s diminue.\n", uke->nom);
125         } break;
126         case SOI_LOW_VIT : {
127             Personnage_stat_mul(uke, STI_VIT, 0.5);
128             printf("La vitesse de %s diminue.\n", uke->nom);
129         } break;
130         case SOI_SOIN : {
131             Personnage_stat_mul(tori, STI_VIE, 1.2);
132             printf("%s se soigne.\n", tori->nom);
133         } break;
134         case SOI_UP_ATK : {
135             Personnage_stat_mul(tori, STI_ATK, 1.4);
136             printf("L'attaque de %s augmente.\n", tori->nom);
137         } break;
138         case SOI_UP_DEF : {
139             Personnage_stat_mul(tori, STI_DEF, 1.2);
140             printf("La defense de %s augmente.\n", tori->nom);
141         } break;
142         case SOI_UP_VIT : {
143             Personnage_stat_mul(tori, STI_VIT, 1.1);
144             printf("La vitesse de %s augmente.\n", tori->nom);
145         } break;
146     default : break;

```

```
146     }
147 }
148
149 int main() {
150     Personnage joueur = Personnage_creer("Joueur", Stats_creer(100,
151         ↪ 50, 20, 100));
152     Personnage adversaire = Personnage_creer("Adversaire",
153         ↪ Stats_creer(100, 50, 20, 100));
154     SortId joueurSort;
155     SortId advSort;
156     int id;
157
158     do {
159         Personnages_afficher(&joueur, &adversaire);
160         afficher_sorts();
161         printf("Votre sort : ");
162         scanf("%d", &id);
163         joueurSort = id;
164         advSort = (rand() % 2 == 1) ? 1 : 1 + (rand() % 8);
165         if(joueur.current.vit >= adversaire.current.vit) {
166             Personnages_appliquer_sort(joueurSort, &joueur,
167                 ↪ &adversaire);
168             Personnages_appliquer_sort(advSort, &adversaire, &joueur);
169         } else {
170             Personnages_appliquer_sort(advSort, &adversaire, &joueur);
171             Personnages_appliquer_sort(joueurSort, &joueur,
172                 ↪ &adversaire);
173         }
174     } while(Personnage_vivant(&joueur) &&
175         ↪ Personnage_vivant(&adversaire));
176
177     if(Personnage_vivant(&joueur)) {
178         printf("Bravo, vous sortez vainqueur du combat.\n");
179     } else {
180         printf("Votre adversaire gagne, essayez encore.\n");
181     }
182
183     exit(EXIT_SUCCESS);
184 }
```

Correction 111 (★ Listes chaînées et ajouts).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Node Node;
5 struct Node {
6     int value;
7     Node * next;
8 };
9
10 Node * Node_alloc(int value) {
11     Node * node = NULL;
12     if((node = (Node *)malloc(sizeof(Node))) == NULL) {
13         fprintf(stderr, "Erreur allocation Node.\n");
14         return NULL;
15     }
16     node->value = value;
17     node->next = NULL;
18     return node;
19 }
20
21 void Node_free(Node ** liste) {
22     if(liste == NULL || *liste == NULL) {
23         return;
24     }
25     Node_free(&(*liste)->next);
26     free(*liste);
27     *liste = NULL;
28 }
29
30 int Node_ajouter(Node ** liste, int value) {
31     Node * current = NULL;
32     if((current = Node_alloc(value)) == NULL) {
33         return 0;
34     }
35     current->next = *liste;
36     *liste = current;
37     return 1;
38 }
39
40 void Node_afficher(const Node * liste) {
```

```
41 printf("[");
42 for(; liste != NULL; liste = liste->next) {
43     printf("%d", liste->value);
44     if(liste->next != NULL) printf(", ");
45 }
46 printf("]\n");
47 }

48 int main() {
49     Node * liste = NULL;
50     int valeurs[] = {1, 2, 3, 4, -1};
51     int i;
52     for(i = 0; valeurs[i] >= 0; ++i) {
53         if(! Node_ajouter(&liste, valeurs[i])) {
54             Node_free(&liste);
55             fprintf(stderr, "Erreur ajout liste : arrêt.\n");
56             exit(EXIT_FAILURE);
57         }
58     }
59     Node_afficher(liste);
60     Node_free(&liste);
61     exit(EXIT_SUCCESS);
62 }
```

Correction 112 (∞ Algorithmes génétiques et problème du voyageur de commerce).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5 #include <SDL/SDL.h>
6 #include <SDL/SDL_gfxPrimitives.h>
7
8 /* Paramètres de la fenêtre : */
9 const int largeur = 800;
10 const int hauteur = 600;
11 const char * titre = "ESGI delivery";
12
13 #define SELECT 10
```

```

14 #define GENERATE 1000
15
16 typedef struct Point Point;
17 struct Point {
18     int x;
19     int y;
20 };
21
22 double Point_distance(const Point * first, const Point * second) {
23     return sqrt((first->x - second->x) * (first->x - second->x) +
24     ↳ (first->y - second->y) * (first->y - second->y));
25 }
26
27 /* Modélisation de la liste utilisateur : une liste de points. */
28 /* Pour permettre un ajout dynamique, nous gardons le nombre */
29 /* de valeurs affectées et la capacité de l'allocation. */
30 Point * userInput = NULL;
31 int taille = 0;
32 int capacite = 0;
33
34 /* Modélisation d'une solution : correspondance avec liste      */
35 /* utilisateur par indices et sauvegarde de la distance de la   */
36 /* solution. */
37 typedef struct Entry Entry;
38 struct Entry {
39     int * indexes;
40     double distance;
41 };
42
43 int generations = 0;
44 Entry * currentEntry = NULL;
45 Entry * entries = NULL;
46 clock_t start;
47
48 /* Initialisation d'une solution sur la liste utilisateur : */
49 Entry Entry_toPositions(Point * points, int taille) {
50     Entry ent;
51     if((ent.indexes = (int *)calloc(taille, sizeof(int))) == NULL) {
52         fprintf(stderr, "Erreur d\\'allocation d\\'une entrée.\n");
53         exit(EXIT_FAILURE);
54 }
```

```
54     ent.distance = 0.;
55     int i;
56     for(i = 0; i < taille; ++i) {
57         if(i) {
58             ent.distance += Point_distance(points + i, points + i - 1);
59         }
60         ent.indexes[i] = i;
61     }
62     return ent;
63 }

64 /* Initialisation de la population de solutions : */
65 void Entries_init() {
66     if((entries = (Entry *)calloc(GENERATE, sizeof(Entry))) == NULL)
67     → {
68         fprintf(stderr, "Erreur d'\allocation de %d entrées
69         → (entrées).\n", GENERATE);
70         exit(EXIT_FAILURE);
71     }
72     int i;
73     for(i = 0; i < GENERATE; ++i) {
74         entries[i] = Entry_toPositions(userInput, taille);
75     }
76     currentEntry = entries;
77 }

78 /* Calcul de la distance pour une solution donnée : */
79 void Entry_compute_distance(Entry * ent) {
80     ent->distance = 0.;
81     int i;
82     for(i = 1; i < taille; ++i) {
83         ent->distance += Point_distance(userInput + ent->indexes[i],
84         → userInput + ent->indexes[i - 1]);
85     }
86 }

87 /* Échange de l'ordre deux positions dans une solution : */
88 void Entries_swap(int i, int j) {
89     if(i == j) {
90         return;
91     }
```

```

92     int k;
93     for(k = 0; k < taille; ++k) {
94         entries[i].indexes[k] ^= entries[j].indexes[k];
95         entries[j].indexes[k] ^= entries[i].indexes[k];
96         entries[i].indexes[k] ^= entries[j].indexes[k];
97     }
98     double d = entries[i].distance;
99     entries[i].distance = entries[j].distance;
100    entries[j].distance = d;
101 }
102
103 void Entries_copy(int dest, int src) {
104     int k;
105     for(k = 0; k < taille; ++k) {
106         entries[dest].indexes[k] = entries[src].indexes[k];
107     }
108     entries[dest].distance = entries[src].distance;
109 }
110
111 /* Mutation : plusieurs échanges sur solution actuelle */
112 void Entry_evolve_lot_of_swaps(Entry * ent) {
113     int psize = rand() % (taille);
114     int k;
115     for(k = 0; k < psize; ++k) {
116         int i;
117         int j;
118         i = rand() % taille;
119         j = rand() % taille;
120         if(i == j)
121             return;
122         ent->indexes[i] ^= ent->indexes[j];
123         ent->indexes[j] ^= ent->indexes[i];
124         ent->indexes[i] ^= ent->indexes[j];
125     }
126 }
127
128 /* Mutation : plusieurs échanges depuis départ */
129 void Entry_evolve_reset_lot_of_swaps(Entry * ent) {
130     int psize = rand() % (taille);
131     int step = rand() % (taille);
132     int k, l, i = 0;

```

```
133     for(l = 0; l < step; ++l) {
134         for(k = l; k < taille; k += step) {
135             ent->indexes[k] = i++;
136         }
137     }
138     for(k = 0; k < psize; ++k) {
139         int i;
140         int j;
141         i = rand() % taille;
142         j = rand() % taille;
143         if(i == j)
144             return;
145         ent->indexes[i] ^= ent->indexes[j];
146         ent->indexes[j] ^= ent->indexes[i];
147         ent->indexes[i] ^= ent->indexes[j];
148     }
149 }
150
151 /* Mutation : un seul échange */
152 void Entry_evolve_unique(Entry * ent) {
153     int i;
154     int j;
155     i = rand() % taille;
156     j = rand() % taille;
157     if(i == j)
158         return;
159     ent->indexes[i] ^= ent->indexes[j];
160     ent->indexes[j] ^= ent->indexes[i];
161     ent->indexes[i] ^= ent->indexes[j];
162 }
163
164 /* Mutation : recherche d'une plus petite distance pour échange
165    ↵ */
166 void Entry_evolve_win_one(Entry * ent) {
167     int i;
168     int j;
169     int psize = rand() % (taille);
170     i = rand() % taille;
171     j = (i + 1) % taille;
172     int k, l;
173     for(k = 0; k < psize; ++k) {
```

```

173     l = rand() % taille;
174     if(l == i)
175         continue;
176     if(Point_distance(userInput + i, userInput + 1) <
177         → Point_distance(userInput + i, userInput + j)) {
178         j = l;
179     }
180     if(i == j)
181         return;
182     ent->indexes[i] ^= ent->indexes[j];
183     ent->indexes[j] ^= ent->indexes[i];
184     ent->indexes[i] ^= ent->indexes[j];
185 }

186 /* Mutation : pivot sur un sous-chemin */
187 void Entry_evolve_propag(Entry * ent) {
188     int i, a, b;
189     int psize = rand() % (taille / 2);
190     i = rand() % taille;
191     int k;
192     for(k = 0; k < psize; ++k) {
193         a = (i + k) % taille;
194         b = (i + 1 + k) % taille;
195         ent->indexes[a] ^= ent->indexes[b];
196         ent->indexes[b] ^= ent->indexes[a];
197         ent->indexes[a] ^= ent->indexes[b];
198     }
199 }
200 }

201 /* Mutation : répétition de recherches minimales */
202 void Entry_evolve_lot_of_win_one(Entry * ent) {
203     int psize = rand() % (taille);
204     int k;
205     for(k = 0; k < psize; ++k) {
206         Entry_evolve_win_one(ent);
207     }
208 }
209 }

210 /* Mutation : tentative de construction d'une chemin minimal */
211 void Entry_evolve_try_chain(Entry * ent) {

```

```
213     int i;
214     int j;
215     int k;
216     int l;
217     int m;
218     i = rand() % taille;
219     j = rand() % taille;
220     for(k = i; k != j; k = (k + 1) % taille) {
221         m = 1;
222         for(l = 2; l < taille; ++l) {
223             if(Point_distance(userInput + k, userInput + ((k + 1) %
224                             → taille)) < Point_distance(userInput + k, userInput + ((k +
225                             → + m) % taille))) {
226                 m = l;
227             }
228             if(m != 1) {
229                 ent->indexes[((k + 1) % taille)] ^= ent->indexes[((k + m) %
230                             → taille)];
231                 ent->indexes[((k + m) % taille)] ^= ent->indexes[((k + 1) %
232                             → taille)];
233                 ent->indexes[((k + 1) % taille)] ^= ent->indexes[((k + m) %
234                             → taille)];
235             }
236         }
237     }
238
239 /* Mutation : permutation de plusieurs positions */
240 void Entry_evolve_perm(Entry * ent) {
241     int perm[20];
242     int psize;
243     if(taille < 20) {
244         psize = rand() % taille;
245     } else {
246         psize = rand() % 20;
247     }
248     if(psize <= 1) {
249         return;
250     }
251     int i;
252     for(i = 0; i < psize; ++i) {
```

```

249     perm[i] = rand() % taille;
250 }
251 for(i = 1; i < psize; ++i) {
252     int a = perm[i], b = perm[0];
253     if(a == b)
254         continue;
255     ent->indexes[a] ^= ent->indexes[b];
256     ent->indexes[b] ^= ent->indexes[a];
257     ent->indexes[a] ^= ent->indexes[b];
258 }
259 }
260
261 /* Mutation : rotation de tout le chemin */
262 void Entry_evolve_rotation(Entry * ent) {
263     int i;
264     for(i = 0; i < taille - 1; ++i) {
265         ent->indexes[i] ^= ent->indexes[i + 1];
266         ent->indexes[i + 1] ^= ent->indexes[i];
267         ent->indexes[i] ^= ent->indexes[i + 1];
268     }
269 }
270
271 /* Mutation : rotations de tout le chemin */
272 void Entry_evolve_rotations(Entry * ent) {
273     int i = rand() % taille;
274     for(; i > 0; --i) {
275         Entry_evolve_rotation(ent);
276     }
277 }
278
279 /* Mutation : mélange aléatoire */
280 void Entry_evolve_shuffle(Entry * ent) {
281     int i, j;
282     for(i = 0; i < taille; ++i) {
283         ent->indexes[i] = -1;
284     }
285     for(i = 0; i < taille; ++i) {
286         for(j = rand() % taille; ent->indexes[j] >= 0; j = (j + 1) %
287             → taille)
288             ;
289         ent->indexes[j] = i;

```

```
289     }
290 }
291
292 /* Mutation d'une solution */
293 void Entry_evolve(Entry * ent) {
294     int choix = 1;
295     /* Selection des fonctions de mutation selon l'avancement : */
296     if(generations > 5000 && taille < 100) {
297         choix = 10;
298     } else if(generations > 2000 && taille < 100) {
299         choix = 6;
300     } else if(generations > 1000 && taille < 100) {
301         choix = 4;
302     } else if(generations > 600) {
303         choix = 3;
304     } else if(generations > 300) {
305         choix = 2;
306     }
307     /* Mélanger aléatoirement la liste utilisateur au début : */
308     if(generations < taille / 10) {
309         Entry_evolve_shuffle(ent);
310         return;
311     }
312     /* Application de la mutation : */
313     switch(rand() % choix) {
314         case 0 :
315             Entry_evolve_unique(ent);
316             break;
317
318         case 1 :
319             Entry_evolve_propag(ent);
320             break;
321
322         case 2 :
323             Entry_evolve_perm(ent);
324             break;
325
326         case 3 :
327             Entry_evolve_rotations(ent);
328             break;
329 }
```

```

330     case 4 :
331         Entry_evolve_win_one(ent);
332         break;
333
334     case 5 :
335         Entry_evolve_try_chain(ent);
336         break;
337
338     case 6 :
339         Entry_evolve_reset_lot_of_swaps(ent);
340         break;
341
342     case 7 :
343         Entry_evolve_lot_of_win_one(ent);
344         break;
345
346     case 8 :
347         Entry_evolve_lot_of_swaps(ent);
348         break;
349
350     case 9 :
351         Entry_evolve_shuffle(ent);
352         break;
353     }
354     /* Pour une génération avancée, combiner des mutations : */
355     if(generations > 500) {
356         int mv = (generations > 5000) ? 5000 : generations;
357         if(rand() % mv > 500) {
358             Entry_evolve(ent);
359         }
360     }
361 }
362
363 /* Duplique et mute dans la population une des meilleures
   → solutions */
364 void Entries_duplicate_best() {
365     int i;
366     for(i = SELECT; i < GENERATE; ++i) {
367         Entries_copy(i, i % SELECT);
368         Entry_evolve(entries + i);
369         Entry_compute_distance(entries + i);

```

```
370     }
371 }
372
373 void Entries_update() {
374     /* Dupliquer et muter les meilleures solutions : */
375     Entries_duplicate_best();
376     int i, j;
377     /* Pour les solutions ayant muté : */
378     for(i = SELECT; i < GENERATE; ++i) {
379         /* Si la solution bat la plus faible de la sélection : */
380         if(entries[i].distance < entries[SELECT - 1].distance) {
381             /* La faire remonter tant qu'elle bat les solutions : */
382             for(j = SELECT - 1; j >= 0; --j) {
383                 if(entries[j].distance < entries[i].distance) {
384                     break;
385                 }
386             }
387             ++j;
388             Entries_swap(i, j);
389         }
390     }
391     currentEntry = entries;
392     ++generations;
393 }

394
395 /* Mise à jour de la capacité de la liste utilisateur : */
396 void updateInput() {
397     if(taille >= capacite) {
398         capacite = taille * 2 + 10;
399         if((userInput = (Point *)realloc(userInput, sizeof(Point) *
400             → capacite)) == NULL) {
401             fprintf(stderr, "Erreur d'\alloc de l'\entrée
402             → utilisateur.\n");
403             exit(EXIT_FAILURE);
404         }
405     }
406
407     /* Ajout d'un point par l'utilisateur : */
408     void addInput(int x, int y) {
409         updateInput();
```

```

409     userInput[taille].x = x;
410     userInput[taille].y = y;
411     ++taille;
412 }
413
414 void affichage(SDL_Surface * ecran) {
415     /* Remplissage de l'écran par un gris foncé uniforme : */
416     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
417             102));
418
419     int i;
420     double distance = 0.;
421     /* Dessin de la liste : */
422     if(currentEntry == NULL) {
423         /* Telle que saisie (avant optimisation) : */
424         for(i = 0; i < taille; ++i) {
425             if(i) {
426                 lineRGBA(ecran, userInput[i - 1].x, userInput[i - 1].y,
427                         userInput[i].x, userInput[i].y, 204, 204, 204, 255);
428                 distance += Point_distance(userInput + i, userInput + i -
429                     1);
430             }
431             filledCircleRGBA(ecran, userInput[i].x, userInput[i].y, 3,
432                             204, 204, 204, 255);
433         }
434     } else {
435         /* Meilleures solution : */
436         for(i = 0; i < taille; ++i) {
437             filledCircleRGBA(ecran, userInput[i].x, userInput[i].y, 3,
438                             204, 204, 204, 255);
439         }
440         for(i = 1; i < taille; ++i) {
441             lineRGBA(ecran, userInput[currentEntry->indexes[i]].x,
442                     userInput[currentEntry->indexes[i]].y,
443                     userInput[currentEntry->indexes[i - 1]].x,
444                     userInput[currentEntry->indexes[i - 1]].y, 204, 204, 0,
445                     255);
446         }
447         distance = currentEntry->distance;
448     }
449     char buffer[512];

```

```
441 sprintf(buffer, "%d positions", taille);
442 stringRGB(A(ecran, 5, 5, buffer, 255, 255, 255, 255));
443 if(currentEntry == NULL) {
444     sprintf(buffer, "Distance totale : %g", distance);
445 } else {
446     sprintf(buffer, "Generation %d (%g s)", generations,
447             (double)(clock() - start) / CLOCKS_PER_SEC);
447 }
448 stringRGB(A(ecran, 5, 25, buffer, 255, 255, 255, 255));
449 if(entries != NULL) {
450     for(i = 0; i < SELECT; ++i) {
451         sprintf(buffer, "Solution %d : %g", i + 1,
452                 entries[i].distance);
452         stringRGB(A(ecran, 5, 45 + i * 20, buffer, 255, 255, 255,
453                     255));
453     }
454 }
455 }
456
457 int Points_save(const char * path, const Point * points, int size)
458 {
459     FILE * file = NULL;
460     if((file = fopen(path, "w+")) == NULL) {
461         fprintf(stderr, "Erreur ouverture %s\n", path);
462         return 0;
463     }
464     int i;
465     fprintf(file, "%d\n", size);
466     for(i = 0; i < size; ++i) {
467         fprintf(file, "%d %d\n", points[i].x, points[i].y);
468     }
469     fclose(file);
470     return 1;
471 }
472
473 int Points_load(const char * path, Point ** points, int * size) {
474     FILE * file = NULL;
475     if((file = fopen(path, "r")) == NULL) {
476         fprintf(stderr, "Erreur ouverture %s\n", path);
477         return 0;
477 }
```

```

478 int i;
479 fscanf(file, "%d", size);
480 if((*points = (Point *)calloc(*size, sizeof(Point))) == NULL) {
481     fprintf(stderr, "Erreur allocation %s\n", path);
482     fclose(file);
483     return 0;
484 }
485 for(i = 0; i < *size; ++i) {
486     fscanf(file, "%d %d", &((*points)[i].x), &((*points)[i].y));
487 }
488 fclose(file);
489 return 1;
490 }

491 int main(int argc, char * argv[]) {
492     srand(time(NULL));
493     /* Création d'une fenêtre SDL : */
494     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
495         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
496         exit(EXIT_FAILURE);
497     }
498     SDL_Surface * ecran = NULL;
499     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
500         | SDL_DOUBLEBUF)) == NULL) {
501         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
502             SDL_GetError());
503         SDL_Quit();
504         exit(EXIT_FAILURE);
505     }
506     SDL_WM_SetCaption(titre, NULL);

507     if(argc > 1) {
508         Points_load(argv[1], &userInput, &taille);
509         capacite = taille;
510     }

511     int active = 1;
512     int optimise = 0;
513     SDL_Event event;

514     while(active) {

```

```
517
518     affichage(ecran);
519     SDL_Flip(ecran);
520
521     while(SDL_PollEvent(&event)) {
522
523         switch(event.type) {
524             /* Utilisateur clique sur la croix de la fenêtre : */
525             case SDL_QUIT : {
526                 active = 0;
527             } break;
528
529             /* Utilisateur enfonce une touche du clavier : */
530             case SDL_KEYDOWN : {
531                 switch(event.key.keysym.sym) {
532                     /* Touche Echap : */
533                     case SDLK_ESCAPE : {
534                         active = 0;
535                     } break;
536
537                     default : break;
538                 }
539             } break;
540
541             case SDL_KEYUP : {
542                 switch(event.key.keysym.sym) {
543                     case SDLK_SPACE : {
544                         if(! optimise) {
545                             Entries_init();
546                             start = clock();
547                         }
548                         optimise = 1;
549                     } break;
550
551                     case SDLK_s : {
552                         if(Points_save("save.points", userInput, taille)) {
553                             printf("Sauvegarde des points dans
554                             ↳ \"save.points\"\n");
555                         }
556                     } break;
557                 }
558             } break;
559         }
560     }
561 }
```

```

557         default : break;
558     }
559 } break;

560
561     case SDL_MOUSEBUTTONUP : {
562         switch(event.button.button) {
563             case SDL_BUTTON_LEFT : {
564                 if(! optimise) {
565                     addInput(event.button.x, event.button.y);
566                 }
567             } break;
568
569             default : break;
570         }
571     } break;

572     default : break;
573 }
574 }
575
576 if(optimise) {
577     Entries_update();
578     SDL_Delay(1);
579 } else {
580     SDL_Delay(1000 / 60);
581 }
582
583 }
584
585     SDL_FreeSurface(ecran);
586     SDL_Quit();
587     exit(EXIT_SUCCESS);
588 }
```

Correction 113 (★★★ Implémenter une liste chaînée).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct LinkedList * LinkedList;
5 struct LinkedList {
```

```
6     int value;
7     LinkedList next;
8 };
9
10    /* Renvoie une liste vide */
11   /* Complexité : O(1) */
12   LinkedList LL_empty();
13
14   /* Libère la liste */
15   /* Complexité : O(n) */
16   void LL_free(LinkedList * liste);
17
18   /* Ajoute un élément en fin */
19   /* Complexité : O(n) */
20   int LL_add_tail(LinkedList * liste, int value);
21
22   /* Ajoute un élément en tête */
23   /* Complexité : O(1) */
24   int LL_add_head(LinkedList * liste, int value);
25
26   /* Ajoute un élément à une position donnée */
27   /* Complexité : O(n) */
28   int LL_insert(LinkedList * liste, int id, int value);
29
30   /* Supprime l'élément en fin */
31   /* Complexité : O(n) */
32   int LL_pop_tail(LinkedList * liste, int * value);
33
34   /* Supprime l'élément en tête */
35   /* Complexité : O(1) */
36   int LL_pop_head(LinkedList * liste, int * value);
37
38   /* Supprime l'élément à une position donnée */
39   /* Complexité : O(n) */
40   int LL_delete(LinkedList * liste, int id, int * value);
41
42   /* Affiche la liste */
43   /* Complexité : O(n) */
44   void LL_print(FILE * flow, const LinkedList * liste);
45
46   typedef struct ArrayList ArrayList;
```

```

47 struct ArrayList {
48     int * values;
49     int size;
50     int capacite;
51 };
52
53 /* Renvoie une liste vide */
54 /* Complexité : O(1) */
55 ArrayList AL_empty();
56
57 /* Libère la liste */
58 /* Complexité : O(1) */
59 void AL_free(ArrayList * liste);
60
61 /* Ajoute un élément en fin */
62 /* Complexité : O(1) */
63 int AL_add_tail(ArrayList * liste, int value);
64
65 /* Ajoute un élément en tête */
66 /* Complexité : O(n) */
67 int AL_add_head(ArrayList * liste, int value);
68
69 /* Ajoute un élément à une position donnée */
70 /* Complexité : O(n) */
71 int AL_insert(ArrayList * liste, int id, int value);
72
73 /* Supprime l'élément en fin */
74 /* Complexité : O(1) */
75 int AL_pop_tail(ArrayList * liste, int * value);
76
77 /* Supprime l'élément en tête */
78 /* Complexité : O(n) */
79 int AL_pop_head(ArrayList * liste, int * value);
80
81 /* Supprime l'élément à une position donnée */
82 /* Complexité : O(n) */
83 int AL_delete(ArrayList * liste, int id, int * value);
84
85 /* Affiche la liste */
86 /* Complexité : O(n) */
87 void AL_print(FILE * flow, const ArrayList * liste);

```

```
88
89 int main() {
90     int v;
91
92     ArrayList al = AL_empty();
93     AL_print(stdout, &al);
94     AL_add_tail(&al, 1);
95     AL_print(stdout, &al);
96     AL_add_tail(&al, 2);
97     AL_print(stdout, &al);
98     AL_add_head(&al, -1);
99     AL_print(stdout, &al);
100    AL_add_head(&al, -2);
101    AL_print(stdout, &al);
102    AL_insert(&al, 2, 0);
103    AL_print(stdout, &al);
104    AL_pop_tail(&al, &v);
105    printf("v : %d; ", v);
106    AL_print(stdout, &al);
107    AL_pop_head(&al, &v);
108    printf("v : %d; ", v);
109    AL_print(stdout, &al);
110    AL_delete(&al, 1, &v);
111    printf("v : %d; ", v);
112    AL_print(stdout, &al);
113    AL_free(&al);
114
115    LinkedList ll = LL_empty();
116    LL_print(stdout, &ll);
117    LL_add_tail(&ll, 1);
118    LL_print(stdout, &ll);
119    LL_add_tail(&ll, 2);
120    LL_print(stdout, &ll);
121    LL_add_head(&ll, -1);
122    LL_print(stdout, &ll);
123    LL_add_head(&ll, -2);
124    LL_print(stdout, &ll);
125    LL_insert(&ll, 2, 0);
126    LL_print(stdout, &ll);
127    LL_pop_tail(&ll, &v);
128    printf("v : %d; ", v);
```

```

129 LL_print(stdout, &ll);
130 LL_pop_head(&ll, &v);
131 printf("v : %d; ", v);
132 LL_print(stdout, &ll);
133 LL_delete(&ll, 1, &v);
134 printf("v : %d; ", v);
135 LL_print(stdout, &ll);
136 LL_free(&ll);

137 exit(EXIT_SUCCESS);
}

140
/* Allocation d'un maillon de la liste */
141 static LinkedList LL_alloc(int value, LinkedList next) {
142     LinkedList res = NULL;
143     if((res = (LinkedList)malloc(sizeof(struct LinkedList))) ==
144         → NULL) {
145         return NULL;
146     }
147     res->value = value;
148     res->next = next;
149     return res;
150 }

151
/* Une liste vide correspond à NULL : aucun maillons */
152 LinkedList LL_empty() {
153     return NULL;
154 }

155
/* Nous libérons récursivement les maillons de la liste */
156 void LL_free(LinkedList * liste) {
157     if(liste == NULL || *liste == NULL) {
158         return;
159     }
160     LL_free(&(*liste)->next);
161     free(*liste);
162 }
163
164
/* Pour ajouter en fin, nous avons besoin de parcourir tous      */
165 /* les éléments de la liste                                     */
166 /* Nous profitons d'avoir un déréférencement pour assigner le */
167 /*                                     */
168

```

```
169 /* nouveau maillon en fin. */  
170 int LL_add_tail(LinkedList * liste, int value) {  
171     for(; *liste != NULL; liste = &((*liste)->next)) ;  
172     if((*liste = LL_alloc(value, NULL)) == NULL) {  
173         return 0;  
174     }  
175     return 1;  
176 }  
177  
178 /* L'ajout en tête se fait instantanément en mettant la liste */  
179 /* courante en fin d'un nouveau maillon. Notre tête de liste */  
180 /* devient ce nouveau maillon. */  
181 int LL_add_head(LinkedList * liste, int value) {  
182     if((*liste = LL_alloc(value, *liste)) == NULL) {  
183         return 0;  
184     }  
185     return 1;  
186 }  
187  
188 /* Nous parcourons autant d'éléments que demandé et insérons */  
189 /* un nouveau maillon à l'emplacement donné. */  
190 int LL_insert(LinkedList * liste, int id, int value) {  
191     for(; *liste != NULL && id > 0; liste = &((*liste)->next), --id)  
192     ;  
193     if(id > 0) {  
194         return 0;  
195     }  
196     if((*liste = LL_alloc(value, *liste)) == NULL) {  
197         return 0;  
198     }  
199     return 1;  
200 }  
201  
202 /* Nous parcourons la liste jusqu'à avoir en main le dernier */  
203 /* maillon. */  
204 int LL_pop_tail(LinkedList * liste, int * value) {  
205     if(*liste == NULL) {  
206         return 0;  
207     }  
208     for(; (*liste)->next != NULL; liste = &((*liste)->next)) ;  
209     *value = (*liste)->value;
```

```

209     free(*liste);
210     *liste = NULL;
211     return 1;
212 }
213
214 /* Nous sauvegardons la tête de liste pour la libérer ensuite. */
215 /* La tête devient l'élément qui suit le premier maillon. */
216 int LL_pop_head(LinkedList * liste, int * value) {
217     if(*liste == NULL) {
218         return 0;
219     }
220     LinkedList current = *liste;
221     *liste = (*liste)->next;
222     *value = current->value;
223     free(current);
224     return 1;
225 }
226
227 int LL_delete(LinkedList * liste, int id, int * value) {
228     if(*liste == NULL) {
229         return 0;
230     }
231     for(; (*liste)->next != NULL && id > 0; liste =
232         → &((*liste)->next), --id) ;
233     if(id > 0) {
234         return 0;
235     }
236     LinkedList current = *liste;
237     *liste = (*liste)->next;
238     *value = current->value;
239     free(current);
240     return 1;
241 }
242 void LL_print(FILE * flow, const LinkedList * liste) {
243     fprintf(flow, "LinkedList : [");
244     int i;
245     for(i = 0; *liste != NULL; ++i, liste = &((*liste)->next)) {
246         if(i) fprintf(flow, ", ");
247         fprintf(flow, "%d", (*liste)->value);
248     }

```

```
249     fprintf(flow, "]\n");
250 }
251
252 static int AL_update(ArrayList * liste) {
253     if(liste->size < liste->capacite) {
254         return 1;
255     }
256     int newCap = liste->size * 2 + 10;
257     int * newValues = NULL;
258     if((newValues = (int *)realloc(liste->values, sizeof(int) *
259         ↳ newCap)) == NULL) {
260         return 0;
261     }
262     liste->capacite = newCap;
263     liste->values = newValues;
264     return 1;
265 }
266
267 ArrayList AL_empty() {
268     ArrayList res;
269     res.values = NULL;
270     res.capacite = 0;
271     res.size = 0;
272     return res;
273 }
274
275 void AL_free(ArrayList * liste) {
276     if(liste == NULL) {
277         return;
278     }
279     if(liste->values != NULL) {
280         free(liste->values);
281         liste->values = NULL;
282     }
283     liste->capacite = 0;
284     liste->size = 0;
285 }
286
287 int AL_add_tail(ArrayList * liste, int value) {
288     if(! AL_update(liste)) {
289         return 0;
```

```
289 }
290     liste->values[(liste->size)++] = value;
291     return 1;
292 }
293
294 int AL_add_head(ArrayList * liste, int value) {
295     if(! AL_update(liste)) {
296         return 0;
297     }
298     int i;
299     for(i = liste->size; i > 0; --i) {
300         liste->values[i] = liste->values[i - 1];
301     }
302     liste->values[0] = value;
303     (liste->size)++;
304     return 1;
305 }
306
307 int AL_insert(ArrayList * liste, int id, int value) {
308     if(! AL_update(liste)) {
309         return 0;
310     }
311     int i;
312     for(i = liste->size; i > id; --i) {
313         liste->values[i] = liste->values[i - 1];
314     }
315     liste->values[id] = value;
316     (liste->size)++;
317     return 1;
318 }
319
320 int AL_pop_tail(ArrayList * liste, int * value) {
321     if(liste->size <= 0) {
322         return 0;
323     }
324     *value = liste->values[--(liste->size)];
325     return 1;
326 }
327
328 int AL_pop_head(ArrayList * liste, int * value) {
329     if(liste->size <= 0) {
```

```
330     return 0;
331 }
332 *value = liste->values[0];
333 int i;
334 for(i = 0; i < liste->size - 1; ++i) {
335     liste->values[i] = liste->values[i + 1];
336 }
337 --(liste->size);
338 return 1;
339 }

340
341 int AL_delete(ArrayList * liste, int id, int * value) {
342     if(liste->size <= 0) {
343         return 0;
344     }
345     *value = liste->values[id];
346     int i;
347     for(i = id; i < liste->size - 1; ++i) {
348         liste->values[i] = liste->values[i + 1];
349     }
350     --(liste->size);
351     return 1;
352 }

353
354 void AL_print(FILE * flow, const ArrayList * liste) {
355     fprintf(flow, "ArrayList : [");
356     int i;
357     for(i = 0; i < liste->size; ++i) {
358         if(i) fprintf(flow, ", ");
359         fprintf(flow, "%d", liste->values[i]);
360     }
361     fprintf(flow, "]\\n");
362 }
```

Correction 114 (∞ Combat de créatures par fichiers).

Version dans un seul fichier :

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```
3 #include <string.h>
4 #include <math.h>
5 #include <time.h>
6 #include <SDL/SDL.h>
7 #include <SDL/SDL_gfxPrimitives.h>
8
9 /* Paramètres de la fenêtre : */
10 const int largeur = 800;
11 const int hauteur = 600;
12 const char * titre = "ESGI fight";
13 SDL_Surface * ecran = NULL;
14
15 typedef struct Stats Stats;
16 struct Stats {
17     int vie;
18     int atk;
19     int def;
20     int vit;
21 };
22
23 typedef enum {
24     S_NONE    = 0,
25     S_LINE    = 1,
26     S_CIRCLE  = 2,
27     S_POLYGON = 3
28 } Shape;
29
30 typedef enum {
31     T_NONE    = 0,
32     T_SELF    = 1,
33     T_OTHER   = 2
34 } Target;
35
36 typedef enum {
37     S_VALUE   = 0,
38     S_VIE    = 1,
39     S_ATK    = 2,
40     S_DEF    = 3,
41     S_VIT    = 4
42 } Stat;
43
```

```
44 typedef enum {
45     O_NONE      = 0,
46     O_EVAL       = 1,
47     O_AFFECT     = 2,
48     O_ADD        = 3,
49     O_AFF_ADD    = 4,
50     O_SUB        = 5,
51     O_AFF_SUB    = 6,
52     O_MUL        = 7,
53     O_AFF_MUL   = 8,
54     O_DIV        = 9,
55     O_AFF_DIV   = 10
56 } Operation;
57
58 typedef struct Action Action;
59 struct Action {
60     Stat stat;
61     Target target;
62     float value;
63     Operation op;
64     Action * left;
65     Action * right;
66     Action * next;
67 };
68
69 int isop(char c) {
70     return c == '+'
71     || c == '-'
72     || c == '*'
73     || c == '/';
74 }
75
76 Operation getop(char c) {
77     switch(c) {
78         case '=' : return O_AFFECT;
79         case '+' : return O_ADD;
80         case '-' : return O_SUB;
81         case '*' : return O_MUL;
82         case '/' : return O_DIV;
83         default : return O_NONE;
84     }
```

```

85 }
86
87 Operation getaffop(char c) {
88     Operation op = getop(c);
89     if(op > O_AFFECT) {
90         return op + 1;
91     }
92     return O_AFFECT;
93 }
94
95 int getprio(char c) {
96     switch(c) {
97         case '=' : return 3;
98         case '+' : return 4;
99         case '-' : return 4;
100        case '*' : return 5;
101        case '/' : return 5;
102        default : return 0;
103    }
104 }
105
106 Target Target_trad(const char * v) {
107     if(strcmp(v, "self") == 0) {
108         return T_SELF;
109     } else if(strcmp(v, "other") == 0) {
110         return T_OTHER;
111     }
112     return T_NONE;
113 }
114
115 Stat Stat_trad(const char * v) {
116     if(strcmp(v, "vie") == 0) {
117         return S_VIE;
118     } else if(strcmp(v, "atk") == 0) {
119         return S_ATK;
120     } else if(strcmp(v, "def") == 0) {
121         return S_DEF;
122     } else if(strcmp(v, "vit") == 0) {
123         return S_VIT;
124     }
125     return S_NONE;

```

```
126 }
127
128 Action * Action_alloc(Action * next) {
129     Action * action = NULL;
130     if((action = (Action *)malloc(sizeof(Action))) == NULL) {
131         return NULL;
132     }
133     action->stat = S_VALUE;
134     action->target = T_NONE;
135     action->op = O_EVAL;
136     action->value = 0.f;
137     action->left = NULL;
138     action->right = NULL;
139     action->next = next;
140     return action;
141 }
142
143 int Action_read_eval(Action ** action, const char * start, const
144 ← char * end) {
145     const char * c;
146     char target[50];
147     char stat[50];
148     if((*action = Action_alloc(*action)) == NULL) {
149         fprintf(stderr, "Erreur allocation action.\n");
150         exit(EXIT_FAILURE);
151     }
152     for(c = start; c != end; ++c) {
153         if(*c == '$') {
154             sscanf(c + 1, "%[^.].%s", target, stat);
155             (*action)->stat = Stat_trad(stat);
156             (*action)->target = Target_trad(target);
157             return 1;
158         }
159         float val;
160         if(sscanf(start, "%f", &val) != 1) {
161             fprintf(stderr, "Erreur evaluation de \"%s\".", start);
162             free(*action);
163             *action = NULL;
164             return 0;
165         }

```

```

166     (*action)->stat = S_VALUE;
167     (*action)->value = val;
168     return 1;
169 }
170
171 int Action_read(Action ** action, const char * start, const char *
172 → end) {
172     int par = 0;
173     const char * op_offset = NULL;
174     int op_prio = -1;
175     const char * c;
176     int has_content = 0;
177     for(c = start; c != end; ++c) {
178         if(*c == '(') {
179             ++par;
180             continue;
181         }
182         if(*c == ')') {
183             --par;
184             continue;
185         }
186         if(*c == ' ' || *c == '\t' || *c == '\n') {
187             continue;
188         }
189         if(par) {
190             continue;
191         }
192         has_content = 1;
193         if(isop(*c) || *c == '=') {
194             int prio = getprio(*c);
195             if(op_prio <= 0 || prio < op_prio) {
196                 op_offset = c;
197                 op_prio = prio;
198             }
199             continue;
200         }
201     }
202     if(! has_content) {
203         for(; *start != '(' && start < end; ++start);
204         for(; *end != ')' && start < end; --end);
205         if(start >= end) {

```

```
206     return 0;
207 }
208 ++start;
209 ++end;
210 return Action_read(action, start, end);
211 }
212 if(op_prio <= 0) {
213     return Action_read_eval(action, start, end);
214 }
215 if((*action = Action_alloc(*action)) == NULL) {
216     fprintf(stderr, "Erreur allocation action.\n");
217     exit(EXIT_FAILURE);
218 }
219 if(*op_offset == '=') {
220     (*action)->op = getaffop(*(op_offset - 1));
221     if(isop(*(op_offset - 1))) {
222         return Action_read(&((*action)->left), start, op_offset - 1)
223             && Action_read(&((*action)->right), op_offset + 1, end);
224     }
225     return Action_read(&((*action)->left), start, op_offset)
226         && Action_read(&((*action)->right), op_offset + 1, end);
227 }
228 (*action)->op = getop(*op_offset);
229 return Action_read(&((*action)->left), start, op_offset)
230     && Action_read(&((*action)->right), op_offset + 1, end);
231 }

232 void Action_debug(const Action * action) {
233     if(! action) {
234         return;
235     }
236     if(action->op == O_EVAL) {
237         if(action->stat == S_VALUE) {
238             printf("%g", action->value);
239             return;
240         }
241         switch(action->target) {
242             case T_SELF : printf("self"); break;
243             case T_OTHER : printf("other"); break;
244             default : break;
245         }
246 }
```

```

247     printf(".");
248     switch(action->stat) {
249         case S_VIE : printf("vie"); break;
250         case S_ATK : printf("atk"); break;
251         case S_DEF : printf("def"); break;
252         case S_VIT : printf("vit"); break;
253         default : break;
254     }
255     return;
256 }
257 printf("(");
258 Action_debug(action->left);
259 switch(action->op) {
260     case O_AFFECT : printf(" = "); break;
261     case O_ADD : printf(" + "); break;
262     case O_SUB : printf(" - "); break;
263     case O_MUL : printf(" * "); break;
264     case O_DIV : printf(" / "); break;
265     case O_AFF_ADD : printf(" += "); break;
266     case O_AFF_SUB : printf(" -= "); break;
267     case O_AFF_MUL : printf(" *= "); break;
268     case O_AFF_DIV : printf(" /= "); break;
269     default : break;
270 }
271 Action_debug(action->right);
272 printf(")");
273 }

274 float Action_eval(const Action * action, Stats * self_stats, Stats
275 → * other_stats) {
276     if(! action) {
277         return 0.f;
278     }
279     if(action->op == O_EVAL) {
280         if(action->stat == S_VALUE) {
281             return action->value;
282         }
283         Stats * look = NULL;
284         switch(action->target) {
285             case T_SELF : look = self_stats; break;
286             case T_OTHER : look = other_stats; break;

```

```
287     default : return 0.f;
288 }
289 switch(action->stat) {
290     case S_VIE : return look->vie;
291     case S_ATK : return look->atk;
292     case S_DEF : return look->def;
293     case S_VIT : return look->vit;
294     default : break;
295 }
296 return 0.f;
297 }
298 if(action->op == O_AFFECT
299 || action->op == O_AFF_ADD
300 || action->op == O_AFF_SUB
301 || action->op == O_AFF_MUL
302 || action->op == O_AFF_DIV) {
303     int * left = NULL;
304     Action * leftNode = action->left;
305     Stats * look = NULL;
306     switch(leftNode->target) {
307         case T_SELF : look = self_stats; break;
308         case T_OTHER : look = other_stats; break;
309         default : return 0.f;
310     }
311     switch(leftNode->stat) {
312         case S_VIE : left = &(look->vie); break;
313         case S_ATK : left = &(look->atk); break;
314         case S_DEF : left = &(look->def); break;
315         case S_VIT : left = &(look->vit); break;
316         default : return 0.f;
317     }
318     float right = Action_eval(action->right, self_stats,
319                               other_stats);
320     switch(action->op) {
321         case O_AFFECT : return (*left) = right;
322         case O_AFF_ADD : return (*left) += right;
323         case O_AFF_SUB : return (*left) -= right;
324         case O_AFF_MUL : return (*left) *= right;
325         case O_AFF_DIV : return (*left) /= right;
326         default : return 0.f;
327 }
```

```

327 } else {
328     float left = Action_eval(action->left, self_stats,
329     ↪ other_stats);
330     float right = Action_eval(action->right, self_stats,
331     ↪ other_stats);
332     switch(action->op) {
333         case O_ADD : return left + right;
334         case O_SUB : return left - right;
335         case O_MUL : return left * right;
336         case O_DIV : return left / right;
337         default : return 0.f;
338     }
339 }
340
341 typedef struct Capacite Capacite;
342 struct Capacite {
343     char nom[64];
344     char message[1024];
345     Action * action;
346 };
347
348 int Capacite_update(Capacite * cap, const char * data) {
349     char buffer[128];
350     if(strlen(data) < 3) {
351         return 1;
352     }
353     sscanf(data, "%s", buffer);
354     if(strcmp(buffer, "action") == 0) {
355         return Action_read(&(cap->action), data + strlen(buffer) + 1,
356         ↪ data + strlen(data));
357     } else if(strcmp(buffer, "message") == 0) {
358         strcpy(cap->message, data + strlen(buffer) + 1);
359     } else if(strcmp(buffer, "nom") == 0) {
360         strcpy(cap->nom, data + strlen(buffer) + 1);
361     }
362     return 1;
363 }
364
365 int Capacite_load(const char * path, Capacite * cap) {
366     FILE * file = NULL;

```

```
365 if((file = fopen(path, "r")) == NULL) {
366     return 0;
367 }
368 char buffer[1024];
369 int i;
370 strcpy(cap->message, "");
371 cap->action = NULL;
372 int car;
373 while((car = fgetc(file)) != EOF) {
374     i = 0;
375     do {
376         buffer[i++] = car;
377         car = fgetc(file);
378     } while(car != EOF && car != '\n');
379     buffer[i] = '\0';
380     Capacite_update(cap, buffer);
381 }
382 fclose(file);
383 return 1;
384 }

385 void Capacite_debug(const Capacite * cap) {
386     printf("Action : {\n");
387     printf(" - nom : %s\n", cap->nom);
388     printf(" - message : %s\n", cap->message);
389     Action * action;
390     for(action = cap->action; action != NULL; action = action->next)
391     {
392         printf(" - action : "); Action_debug(action); printf("\n");
393     }
394     printf("}\n");
395 }

396 typedef struct Personnage Personnage;
397 struct Personnage {
398     char name[50];
399     Stats start;
400     Stats current;
401     unsigned char face[1024];
402     unsigned char back[1024];
403     Capacite capacites[4];
404 }
```

```

405     int nb_caps;
406 };
407
408 Personnage joueur;
409 Personnage adversaire;
410
411 Personnage Personnage_creer(const char * name, Stats stats) {
412     Personnage perso;
413     strcpy(perso.name, name);
414     perso.start = stats;
415     perso.current = stats;
416     return perso;
417 }
418
419 void barre_vie(int sx, int sy, int ex, int ey, int value, int
→ max_value) {
420     roundedBoxRGBA(ecran, sx, sy, ex, ey, 3, 51, 51, 51, 255);
421     int i;
422     for(i = 1; i < 4; ++i) {
423         vlineRGBA(ecran, sx + (i / 4.) * (ex - sx), sy, ey, 102, 102,
→ 102, 255);
424     }
425     if(value <= 0) {
426         return;
427     }
428     float t = (float)value / max_value;
429     float r, g, b;
430     if(t > 0.6) {
431         r = 0.; g = 1.; b = 0.;
432     } else if(t > 0.3) {
433         r = (0.6 - t) / 0.3; g = 1.; b = 0. ;
434     } else {
435         r = 1.; g = t / 0.3; b = 0. ;
436     }
437     roundedBoxRGBA(ecran, sx, sy, sx + t * (ex - sx), ey, 3, 51 +
→ 153 * r, 51 + 153 * g, 51 + 153 * b, 255);
438     if(t * (ex - sx) > 6) {
439         roundedBoxRGBA(ecran, sx + 3, sy + 3, sx + t * (ex - sx) - 3,
→ ey - 3, 2, 51 + 102 * r, 51 + 102 * g, 51 + 102 * b, 255);
440     }
441     char buffer[100];

```

```
442     sprintf(buffer, "%d / %d", value, max_value);
443     stringRGBA(ecran, sx + 15, sy + 2, buffer, 255, 255, 255, 255);
444 }
445
446 void barre_stat(int sx, int sy, int ex, int ey, int value, int
447 → start_value, float r, float g, float b) {
448     roundedBoxRGBA(ecran, sx, sy, ex, ey, 3, 102, 102, 102, 255);
449     int i;
450     for(i = 1; i < 6; ++i) {
451         vlineRGBA(ecran, sx + (pow(2., i) / pow(2., 6.)) * (ex - sx),
452 → sy, ey, 153, 153, 153, 255);
453     }
454     start_value *= 4;
455     if(value > start_value) {
456         value = start_value;
457     }
458     float t = (float)value / start_value;
459     roundedBoxRGBA(ecran, sx + t * (ex - sx), ey, 3, 51 +
460 → 153 * r, 51 + 153 * g, 51 + 153 * b, 255);
461     if(t * (ex - sx) > 6) {
462         roundedBoxRGBA(ecran, sx + 3, sy + 3, sx + t * (ex - sx) - 3,
463 → ey - 3, 2, 51 + 102 * r, 51 + 102 * g, 51 + 102 * b, 255);
464     }
465     char buffer[100];
466     sprintf(buffer, "%d", value);
467     stringRGBA(ecran, sx + 15, sy + 2, buffer, 255, 255, 255, 255);
468 }
469
470 void Personnage_afficher_stats(const Personnage * perso, int x,
471 → int y) {
472     int sx = x - largeur / 4;
473     int sy = y - hauteur / 9;
474     int ex = x + largeur / 4;
475     int ey = y + hauteur / 13;
476     roundedBoxRGBA(ecran, sx, sy, ex, ey, 20, 255, 255, 255, 255);
477     roundedBoxRGBA(ecran, sx + 5, sy + 5, ex - 5, ey - 5, 15, 204,
478 → 204, 204, 255);
479     stringRGBA(ecran, sx + 15, sy + 15, perso->name, 51, 51, 51,
480 → 255);
481     hlineRGBA(ecran, sx + 25, ex - 25, sy + 28, 51, 51, 51, 255);
482     stringRGBA(ecran, sx + 15, sy + 35, "Vie : ", 51, 51, 51, 255);
483 }
```

```

476     barre_vie(sx + 60, sy + 32, ex - 15, sy + 44,
477                 ↳ perso->current.vie, perso->start.vie);
478     stringRGB4(écran, sx + 15, sy + 50, "Attaque : ", 51, 51, 51,
479                 ↳ 255);
480     barre_stat(sx + 100, sy + 47, ex - 15, sy + 59,
481                 ↳ perso->current.atk, perso->start.atk, 1., 0., 0.);
482     stringRGB4(écran, sx + 15, sy + 65, "Defense : ", 51, 51, 51,
483                 ↳ 255);
484     barre_stat(sx + 100, sy + 62, ex - 15, sy + 74,
485                 ↳ perso->current.def, perso->start.def, 0., 0., 1.);
486     stringRGB4(écran, sx + 15, sy + 80, "Vitesse : ", 51, 51, 51,
487                 ↳ 255);
488     barre_stat(sx + 100, sy + 77, ex - 15, sy + 89,
489                 ↳ perso->current.vit, perso->start.vit, 1., 1., 0.);
490 }
491
492 void draw_data(unsigned char * data, int cx, int cy, int s) {
493     while(*data) {
494         Shape shape = *(data++);
495         switch(shape) {
496             case S_CIRCLE : {
497                 int x, y, d;
498                 int r, g, b, a;
499                 x = (char)(*data++);
500                 y = (char)(*data++);
501                 d = *(data++);
502                 r = *(data++);
503                 g = *(data++);
504                 b = *(data++);
505                 a = *(data++);
506                 x = cx + x * s / 100.;
507                 y = cy + y * s / 100.;
508                 d = d * s / 100.;
509                 filledCircleRGB4(écran, x, y, d, r, g, b, a);
510             } break;
511
512             case S_LINE : {
513                 int sx, sy, ex, ey, d;
514                 int r, g, b, a;
515                 sx = (char)(*data++);
516                 sy = (char)(*data++);
517             }
518         }
519     }
520 }

```

```
510     ex = (char)(*(data++));
511     ey = (char)(*(data++));
512     d = *(data++);
513     r = *(data++);
514     g = *(data++);
515     b = *(data++);
516     a = *(data++);
517     sx = cx + sx * s / 100.;
518     sy = cy + sy * s / 100.;
519     ex = cx + ex * s / 100.;
520     ey = cy + ey * s / 100.;
521     d = d * s / 100.;
522     thickLineRGBA(ecran, sx, sy, ex, ey, d, r, g, b, a);
523 } break;

524 case S_POLYGON : {
525     Sint16 xs[32];
526     Sint16 ys[32];
527     int n, i;
528     int r, g, b, a;
529     n = *(data++);
530     for(i = 0; i < n; ++i) {
531         xs[i] = (char)(*(data++));
532         ys[i] = (char)(*(data++));
533         xs[i] = cx + xs[i] * s / 100.;
534         ys[i] = cy + ys[i] * s / 100.;
535     }
536     r = *(data++);
537     g = *(data++);
538     b = *(data++);
539     a = *(data++);
540     filledPolygonRGBA(ecran, xs, ys, n, r, g, b, a);
541 } break;

542     default : break;
543 }
544 }
545 }
546 }
547 }

548 int data_load(FILE * fich, unsigned char * draw) {
549     char buffer[50];
```

```

551 for(;;) {
552     fscanf(fich, "%s", buffer);
553     if(strcmp(buffer, "}") == 0) {
554         break;
555     } else if(strcmp(buffer, "circle") == 0) {
556         int x, y, d;
557         int r, g, b, a;
558         if(fscanf(fich, "%d, %d, %d rgba %d, %d, %d, %d\n", &x, &y,
559             &d, &r, &g, &b, &a) != 7) {
560             fprintf(stderr, "Erreur lecture cercle\n");
561             return 0;
562         }
563         *(draw++) = (unsigned char)S_CIRCLE;
564         *(draw++) = (unsigned char)x;
565         *(draw++) = (unsigned char)y;
566         *(draw++) = (unsigned char)d;
567         *(draw++) = (unsigned char)r;
568         *(draw++) = (unsigned char)g;
569         *(draw++) = (unsigned char)b;
570         *(draw++) = (unsigned char)a;
571     } else if(strcmp(buffer, "line") == 0) {
572         int sx, sy, ex, ey, d;
573         int r, g, b, a;
574         if(fscanf(fich, "%d, %d, %d, %d, %d rgba %d, %d, %d, %d\n",
575             &sx, &sy, &ex, &ey, &d, &r, &g, &b, &a) != 9) {
576             fprintf(stderr, "Erreur lecture line\n");
577             return 0;
578         }
579         *(draw++) = (unsigned char)S_LINE;
580         *(draw++) = (unsigned char)sx;
581         *(draw++) = (unsigned char)sy;
582         *(draw++) = (unsigned char)ex;
583         *(draw++) = (unsigned char)ey;
584         *(draw++) = (unsigned char)d;
585         *(draw++) = (unsigned char)r;
586         *(draw++) = (unsigned char)g;
587         *(draw++) = (unsigned char)b;
588         *(draw++) = (unsigned char)a;
589     } else if(strcmp(buffer, "polygon") == 0) {
590         Sint16 xs[32];
591         Sint16 ys[32];

```

```
590     int n, x, y, i;
591     int r, g, b, a;
592     if(fscanf(fich, "%d %d, %d", &n, &x, &y) != 3) {
593         fprintf(stderr, "Erreur lecture polygon : init\n");
594         return 0;
595     }
596     xs[0] = x;
597     ys[0] = y;
598     for(i = 1; i < n; ++i) {
599         if(fscanf(fich, ", %d, %d", &x, &y) != 2) {
600             fprintf(stderr, "Erreur lecture polygon : vertex %d\n",
601                     i);
602             return 0;
603         }
604         xs[i] = x;
605         ys[i] = y;
606     }
607     if(fscanf(fich, " rgba %d, %d, %d, %d\n", &r, &g, &b, &a) !=
608         4) {
609         fprintf(stderr, "Erreur lecture polygon : color\n");
610         return 0;
611     }
612     *(draw++) = (unsigned char)S_POLYGON;
613     *(draw++) = (unsigned char)n;
614     for(i = 0; i < n; ++i) {
615         *(draw++) = (unsigned char)(xs[i]);
616         *(draw++) = (unsigned char)(ys[i]);
617     }
618     *(draw++) = (unsigned char)r;
619     *(draw++) = (unsigned char)g;
620     *(draw++) = (unsigned char)b;
621     *(draw++) = (unsigned char)a;
622 }
623 *draw = (unsigned char)S_NONE;
624 return 1;
625 }
626 int cap_load(FILE * fich, Personnage * perso) {
627     char buffer[50];
628     int i = 0;
```

```

629 for(;;) {
630     fscanf(fich, "%s", buffer);
631     if(strcmp(buffer, "}") == 0) {
632         break;
633     } else if(strcmp(buffer, "load") == 0) {
634         fscanf(fich, "%s", buffer);
635         if(! Capacite_load(buffer, perso->capacites + i)) {
636             fprintf(stderr, "Erreur lecture capacite \"%s\"", buffer);
637         }
638         Capacite_debug(perso->capacites + i);
639         ++i;
640     }
641 }
642 perso->nb_caps = i;
643 return 1;
644 }

645

646 int Personnage_load(Personnage * perso, const char * path) {
647     FILE * fich = NULL;
648     if((fich = fopen(path, "r")) == NULL) {
649         fprintf(stderr, "Erreur lecture d'un personnage \"%s\"\n",
650                 path);
651         return 0;
652     }
653     if(fscanf(fich, "name : %s\n", perso->name) != 1
654     || fscanf(fich, "vie : %d\n", &(perso->start.vie)) != 1
655     || fscanf(fich, "atk : %d\n", &(perso->start.atk)) != 1
656     || fscanf(fich, "def : %d\n", &(perso->start.def)) != 1
657     || fscanf(fich, "vit : %d\n", &(perso->start.vit)) != 1) {
658         fprintf(stderr, "Erreur de lecture des stats pour \"%s\"\n",
659                 path);
660         goto error;
661     }
662     perso->current = perso->start;
663
664     unsigned char * draw = perso->face;
665     fscanf(fich, "face : {\n");
666
667     if(! data_load(fich, draw)) {
668         fprintf(stderr, "Erreur lecture dessin de face pour \"%s\"\n",
669                 path);

```

```
667     goto error;
668 }
669
670 draw = perso->back;
671 fscanf(fich, "back : {\n");
672
673 if(! data_load(fich, draw)) {
674     fprintf(stderr, "Erreur lecture dessin de dos pour \"%s\"\n",
675             ↪ path);
676     goto error;
677 }
678
679 fscanf(fich, "capacities : {\n");
680
681 if(! cap_load(fich, perso)) {
682     fprintf(stderr, "Erreur lecture dessin de dos pour \"%s\"\n",
683             ↪ path);
684     goto error;
685 }
686
687 fclose(fich);
688
689 printf("\\"%s\\" chargé.\n", path);
690 return 1;
691
692 error :
693 fclose(fich);
694 return 0;
695 }
696
697 void affichage() {
698     /* Remplissage de l'écran par un gris foncé uniforme : */
699     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
700                  ↪ 102));
701
702     Personnage_afficher_stats(&adversaire, 2 * largeur / 7, hauteur
703             ↪ / 5);
704     Personnage_afficher_stats(&joueur, 5 * largeur / 7, 3 * hauteur
705             ↪ / 5);
706     draw_data(joueur.back, 3 * largeur / 14, 3 * hauteur / 5,
707             ↪ hauteur / 3);
```

```

702     draw_data(adversaire.face, 11 * largeur / 14, hauteur / 5,
703     ↪ hauteur / 3);
704 }
705
706 int afficher_choix(int mx, int my, const Personnage * perso) {
707     roundedBoxRGBA(ecran, 0, 3 * hauteur / 4, largeur, hauteur, 15,
708     ↪ 204, 204, 204, 255);
709
710     int cap = -1;
711     int i, x, y;
712     if(mx > 0 && mx < largeur && my > 3 * hauteur / 4 && my <
713     ↪ hauteur) {
714         x = mx / (largeur / 2);
715         y = (my - 3 * hauteur / 4) / (hauteur / 8);
716         cap = 2 * y + x;
717         if(cap < perso->nb_caps) {
718             roundedBoxRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 + y *
719             ↪ hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 + (y
720             ↪ + 1) * hauteur / 8, 15, 255, 255, 255);
721         } else {
722             cap = -1;
723         }
724     }
725     for(i = 0; i < perso->nb_caps; ++i) {
726         x = i % 2;
727         y = i / 2;
728         roundedRectangleRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 +
729         ↪ y * hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 +
730         ↪ (y + 1) * hauteur / 8, 15, 102, 102, 102, 255);
731         stringRGBA(ecran, (x + 0.25) * largeur / 2, 3 * hauteur / 4 +
732         ↪ (y + 0.5) * hauteur / 8, perso->capacites[i].nom, 51, 51,
733         ↪ 51, 255);
734     }
735     return cap;
736 }
737
738 void afficher_message(const char * message, const Personnage *
739     ↪ self, const Personnage * other) {
740     char buffer[256] = "";
741     char target[64];
742     char * current = NULL;

```

```
733     for(current = buffer; *message != '\0'; ++message) {
734         if(*message != '$') {
735             *current = *message;
736             ++current;
737             continue;
738         }
739         sscanf(message + 1, "%s", target);
740         if(strcmp(target, "self") == 0) {
741             strcpy(current, self->name);
742             current += strlen(current);
743             message += strlen(target);
744         } else if(strcmp(target, "other") == 0) {
745             strcpy(current, other->name);
746             current += strlen(current);
747             message += strlen(target);
748         }
749     }
750     *current = '\0';
751     roundedBoxRGBA(ecran, 0, 3 * hauteur / 4, largeur, hauteur, 15,
752     ↪ 204, 204, 204, 255);
753     stringRGBA(ecran, largeur / 8, 7 * hauteur / 8, buffer, 51, 51,
754     ↪ 51, 255);
755 }
756
757 void applyCapacite(const Capacite * cap, const Personnage * self,
758     ↪ const Personnage * other, Stats * self_stats, Stats *
759     ↪ other_stats) {
760     *self_stats = self->current;
761     *other_stats = other->current;
762     Action * action = NULL;
763     for(action = cap->action; action != NULL; action = action->next)
764     ↪ {
765         Action_eval(action, self_stats, other_stats);
766     }
767     if(self_stats->vie < 0) self_stats->vie = 0;
768     if(self_stats->atk < self->start.atk / 4) self_stats->atk =
769     ↪ self->start.atk / 4;
770     if(self_stats->def < self->start.def / 4) self_stats->def =
771     ↪ self->start.def / 4;
772     if(self_stats->vit < self->start.vit / 4) self_stats->vit =
773     ↪ self->start.vit / 4;
```

```

766     if(self_stats->vie > self->start.vie) self_stats->vie =
767         → self->start.vie;
768     if(self_stats->atk > self->start.atk * 4) self_stats->atk =
769         → self->start.atk * 4;
770     if(self_stats->def > self->start.def * 4) self_stats->def =
771         → self->start.def * 4;
772     if(self_stats->vit > self->start.vit * 4) self_stats->vit =
773         → self->start.vit * 4;
774     if(other_stats->vie < 0) other_stats->vie = 0;
775     if(other_stats->atk < other->start.atk / 4) other_stats->atk =
776         → other->start.atk / 4;
777     if(other_stats->def < other->start.def / 4) other_stats->def =
778         → other->start.def / 4;
779     if(other_stats->vit < other->start.vit / 4) other_stats->vit =
780         → other->start.vit / 4;
781     if(other_stats->vie > other->start.vie) other_stats->vie =
782         → other->start.vie;
783     if(other_stats->atk > other->start.atk * 4) other_stats->atk =
784         → other->start.atk * 4;
785     if(other_stats->def > other->start.def * 4) other_stats->def =
786         → other->start.def * 4;
787     if(other_stats->vit > other->start.vit * 4) other_stats->vit =
788         → other->start.vit * 4;
789 }
790
791 Stats Stats_interpolate(Stats first, Stats second, float t) {
792     return (Stats){
793         .vie = first.vie * (1 - t) + second.vie * t,
794         .atk = first.atk * (1 - t) + second.atk * t,
795         .def = first.def * (1 - t) + second.def * t,
796         .vit = first.vit * (1 - t) + second.vit * t
797     };
798 }
799
800 void player_turn(int cap) {
801     Stats jc, js, ac, as;
802     jc = joueur.current;
803     ac = adversaire.current;
804     applyCapacite(joueur.capacites + cap, &joueur, &adversaire, &js,
805         → &as);
806     int i;

```

```
795     for(i = 0; i < 20; ++i) {
796         float t = i / 19.;
797         joueur.current = Stats_interpolate(jc, js, t);
798         adversaire.current = Stats_interpolate(ac, as, t);
799         affichage();
800         afficher_message(joueur.capacites[cap].message, &joueur,
801                         → &adversaire);
802         SDL_Flip(ecran);
803         SDL_Delay(100);
804     }
805 }
806
807 void opponent_turn() {
808     Stats jc, js, ac, as;
809     int other_cap = rand() % 2;
810     if(other_cap == 1) {
811         other_cap += rand() % 3;
812     }
813
814     jc = joueur.current;
815     ac = adversaire.current;
816     applyCapacite(adversaire.capacites + other_cap, &adversaire,
817                   → &joueur, &as, &js);
818     int i;
819     for(i = 0; i < 20; ++i) {
820         float t = i / 19.;
821         joueur.current = Stats_interpolate(jc, js, t);
822         adversaire.current = Stats_interpolate(ac, as, t);
823         affichage();
824         afficher_message(adversaire.capacites[other_cap].message,
825                         → &adversaire, &joueur);
826         SDL_Flip(ecran);
827         SDL_Delay(100);
828     }
829 }
830
831 void display_winner() {
832     char buffer[1024];
833     if(adversaire.current.vie <= 0) {
834         roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur /
835                         → 8, 3 * hauteur / 4, 15, 0, 53, 0, 204);
```

```

832     sprintf(buffer, "Victoire de %s", joueur.name);
833 } else {
834     roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur /
835     → 8, 3 * hauteur / 4, 15, 53, 0, 0, 204);
836     sprintf(buffer, "Defaite de %s", joueur.name);
837 }
838 stringRGBA(ecran, 2 * largeur / 8, hauteur / 2, buffer, 255,
839     → 255, 255, 255);
840 }
841
842 int finished() {
843     return joueur.current.vie <= 0 || adversaire.current.vie <= 0;
844 }
845
846 int main() {
847     srand(time(NULL));
848     /* Cr ation d'une fen tre SDL : */
849     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
850         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
851         exit(EXIT_FAILURE);
852     }
853     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
854     → | SDL_DOUBLEBUF)) == NULL) {
855         fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
856         → SDL_GetError());
857         SDL_Quit();
858         exit(EXIT_FAILURE);
859     }
860     SDL_WM_SetCaption(titre, NULL);
861
862     int active = 1;
863     SDL_Event event;
864
865     if(! Personnage_load(&joueur, "first.perso")
866     || ! Personnage_load(&adversaire, "first.perso")) {
867         goto end;
868     }
869
870     int last_mouse_x = 0;
871     int last_mouse_y = 0;
872     int cap = -1;

```

```
869 int select_cap = 0;
870
871 while(active) {
872
873     affichage();
874     cap = afficher_choix(last_mouse_x, last_mouse_y, &joueur);
875     SDL_Flip(ecran);
876
877     while(SDL_PollEvent(&event)) {
878
879         switch(event.type) {
880             /* Utilisateur clique sur la croix de la fenêtre : */
881             case SDL_QUIT : {
882                 active = 0;
883             } break;
884
885             /* Utilisateur enfonce une touche du clavier : */
886             case SDL_KEYDOWN : {
887                 switch(event.key.keysym.sym) {
888                     /* Touche Echap : */
889                     case SDLK_ESCAPE : {
890                         active = 0;
891                     } break;
892
893                     default : break;
894                 }
895             } break;
896
897             case SDL_MOUSEMOTION : {
898                 last_mouse_x = event.motion.x;
899                 last_mouse_y = event.motion.y;
900             } break;
901
902             case SDL_MOUSEBUTTONUP : {
903                 select_cap = 1;
904             } break;
905
906             default : break;
907         }
908     }
909 }
```

```

910     if(select_cap && cap >= 0 && cap < joueur.nb_caps) {
911         if(joueur.current.vit >= adversaire.current.vit) {
912             player_turn(cap);
913             opponent_turn();
914         } else {
915             opponent_turn();
916             player_turn(cap);
917         }
918         active = ! finished();
919
920         select_cap = 0;
921
922         if(! active) {
923             display_winner();
924             SDL_Flip(ecran);
925             SDL_Delay(2000);
926         }
927     }
928
929     SDL_Delay(1000 / 60);
930 }
931
932 end:
933
934     SDL_FreeSurface(ecran);
935     SDL_Quit();
936     exit(EXIT_SUCCESS);
937 }
```

Version modularisée :

```

1 CC= gcc
2 CFLAGS= -O2 -Wall -Werror -ansi -Wno-unused-result
3 CLIBS= -lm -lSDL -lSDL_gfx
4 EXE= executable
5 OBJ= obj/
6 SRC= src/
7 INCL= include/
8 FILEC:= $(wildcard $(SRC)*.c)
9 FILEH:= $(wildcard $(INCL)*.h)
```

```
10 FILEO:= $(patsubst $(SRC)%.c,$(OBJ)%.o,$(FILEC))
11
12 $(EXE) : $(FILEO)
13     $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
14
15 $(OBJ)main.o : $(SRC)main.c $(FILEH)
16     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
17     $(CC) $(CFLAGS) -o $@ -c $<
18
19 $(OBJ)%.o : $(SRC)%.c $(INCL)%.h
20     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
21     $(CC) $(CFLAGS) -o $@ -c $<
22
23 clean :
24     rm -rf $(OBJ)*.o
25     rm -rf $(OBJ)
26     rm -rf $(EXE)
```

```
1 /* +-----+ */
2 /* / FICHIER : src/main.c / */
3 /* +-----+ */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <math.h>
9 #include <time.h>
10
11 #include "../include/personnage.h"
12 #include "../include/game.h"
13
14 int main() {
15     srand(time(NULL));
16     /* Création d'une fenêtre SDL : */
17     if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
18         fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
19         exit(EXIT_FAILURE);
20     }
21     if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
22         | SDL_DOUBLEBUF)) == NULL) {
```



```
62         default : break;
63     }
64 } break;

66
67 case SDL_MOUSEMOTION : {
68     last_mouse_x = event.motion.x;
69     last_mouse_y = event.motion.y;
70 } break;

71
72 case SDL_MOUSEBUTTONDOWN : {
73     select_cap = 1;
74 } break;

75
76     default : break;
77 }
78 }

79 if(select_cap && cap >= 0 && cap < joueur.nb_caps) {
80     if(joueur.current.vit >= adversaire.current.vit) {
81         player_turn(cap);
82         opponent_turn();
83     } else {
84         opponent_turn();
85         player_turn(cap);
86     }
87     active = ! finished();
88
89     select_cap = 0;
90
91     if(! active) {
92         display_winner();
93         SDL_Flip(ecran);
94         SDL_Delay(2000);
95     }
96 }
97

98     SDL_Delay(1000 / 60);
99 }
100

101 end:
```

```

103     SDL_FreeSurface(ecran);
104     SDL_Quit();
105     exit(EXIT_SUCCESS);
106 }
107

```

```

1  /* +-----+ */
2  /* / FICHIER : include/game.h | */
3  /* +-----+ */

4
5 #ifndef DEF_HEADER_ESGI_FIGHT_GAME
6 #define DEF_HEADER_ESGI_FIGHT_GAME

7
8 #include "personnage.h"
9 #include "capacite.h"
10 #include "stats.h"
11 #include "draw.h"

12
13 extern Personnage joueur;
14 extern Personnage adversaire;

15
16 void affichage();

17
18 int afficher_choix(int mx, int my, const Personnage * perso);

19
20 void afficher_message(const char * message, const Personnage *
→ self, const Personnage * other);

21
22 void applyCapacite(const Capacite * cap, const Personnage * self,
→ const Personnage * other, Stats * self_stats, Stats *
→ other_stats);

23
24 void player_turn(int cap);

25
26 void opponent_turn();

27
28 void display_winner();

29
30 int finished();
31

```

```
32 #endif
```

```
1  /* +-----+ */
2  /* / FICHIER : src/game.c | */
3  /* +-----+ */
4
5  #include "../include/game.h"
6
7  #include <stdio.h>
8  #include <string.h>
9  #include <SDL/SDL_gfxPrimitives.h>
10
11 Personnage joueur;
12 Personnage adversaire;
13
14 void affichage() {
15     /* Remplissage de l'écran par un gris foncé uniforme : */
16     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 51, 51,
17         ↵ 102));
18
19     Personnage_afficher_stats(&adversaire, 2 * largeur / 7, hauteur
20         ↵ / 5);
21     Personnage_afficher_stats(&joueur, 5 * largeur / 7, 3 * hauteur
22         ↵ / 5);
23     draw_data(joueur.back, 3 * largeur / 14, 3 * hauteur / 5,
24         ↵ hauteur / 3);
25     draw_data(adversaire.face, 11 * largeur / 14, hauteur / 5,
26         ↵ hauteur / 3);
27 }
28
29 int afficher_choix(int mx, int my, const Personnage * perso) {
30     roundedBoxRGBA(ecran, 0, 3 * hauteur / 4, largeur, hauteur, 15,
31         ↵ 204, 204, 204, 255);
32
33     int cap = -1;
34     int i, x, y;
35     if(mx > 0 && mx < largeur && my > 3 * hauteur / 4 && my <
36         ↵ hauteur) {
37         x = mx / (largeur / 2);
38         y = (my - 3 * hauteur / 4) / (hauteur / 8);
```

```

32     cap = 2 * y + x;
33     if(cap < perso->nb_caps) {
34         roundedBoxRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 + y *
35             hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 + (y
36             + 1) * hauteur / 8, 15, 255, 255, 255);
37     } else {
38         cap = -1;
39     }
40 }
41 for(i = 0; i < perso->nb_caps; ++i) {
42     x = i % 2;
43     y = i / 2;
44     roundedRectangleRGBA(ecran, x * largeur / 2, 3 * hauteur / 4 +
45         y * hauteur / 8, (x + 1) * largeur / 2, 3 * hauteur / 4 +
46         (y + 1) * hauteur / 8, 15, 102, 102, 102, 255);
47     stringRGBA(ecran, (x + 0.25) * largeur / 2, 3 * hauteur / 4 +
48         (y + 0.5) * hauteur / 8, perso->capacites[i].nom, 51, 51,
49         51, 255);
50 }
51 return cap;
52 }

53 void afficher_message(const char * message, const Personnage *
54     self, const Personnage * other) {
55     char buffer[256] = "";
56     char target[64];
57     char * current = NULL;
58     for(current = buffer; *message != '\0'; ++message) {
59         if(*message != '$') {
60             *current = *message;
61             ++current;
62             continue;
63         }
64         sscanf(message + 1, "%s", target);
65         if(strcmp(target, "self") == 0) {
66             strcpy(current, self->name);
67             current += strlen(current);
68             message += strlen(target);
69         } else if(strcmp(target, "other") == 0) {
70             strcpy(current, other->name);
71             current += strlen(current);
72         }
73     }
74 }
```

```
66         message += strlen(target);
67     }
68 }
69 *current = '\0';
70 roundedBoxRGBA(ecran, 0, 3 * hauteur / 4, largeur, hauteur, 15,
71   ↪ 204, 204, 204, 255);
72 stringRGBA(ecran, largeur / 8, 7 * hauteur / 8, buffer, 51, 51,
73   ↪ 51, 255);
74 }
75
76 void applyCapacite(const Capacite * cap, const Personnage * self,
77   ↪ const Personnage * other, Stats * self_stats, Stats *
78   ↪ other_stats) {
79     *self_stats = self->current;
80     *other_stats = other->current;
81     Action * action = NULL;
82     for(action = cap->action; action != NULL; action = action->next)
83     {
84       Action_eval(action, self_stats, other_stats);
85     }
86     if(self_stats->vie < 0) self_stats->vie = 0;
87     if(self_stats->atk < self->start.atk / 4) self_stats->atk =
88       ↪ self->start.atk / 4;
89     if(self_stats->def < self->start.def / 4) self_stats->def =
90       ↪ self->start.def / 4;
91     if(self_stats->vit < self->start.vit / 4) self_stats->vit =
92       ↪ self->start.vit / 4;
93     if(self_stats->vie > self->start.vie) self_stats->vie =
94       ↪ self->start.vie;
95     if(self_stats->atk > self->start.atk * 4) self_stats->atk =
96       ↪ self->start.atk * 4;
97     if(self_stats->def > self->start.def * 4) self_stats->def =
98       ↪ self->start.def * 4;
99     if(self_stats->vit > self->start.vit * 4) self_stats->vit =
100       ↪ self->start.vit * 4;
101     if(other_stats->vie < 0) other_stats->vie = 0;
102     if(other_stats->atk < other->start.atk / 4) other_stats->atk =
103       ↪ other->start.atk / 4;
104     if(other_stats->def < other->start.def / 4) other_stats->def =
105       ↪ other->start.def / 4;
106     if(other_stats->vit < other->start.vit / 4) other_stats->vit =
107       ↪ other->start.vit / 4;
```

```

93     if(other_stats->vie > other->start.vie) other_stats->vie =
94         ↪ other->start.vie;
95     if(other_stats->atk > other->start.atk * 4) other_stats->atk =
96         ↪ other->start.atk * 4;
97     if(other_stats->def > other->start.def * 4) other_stats->def =
98         ↪ other->start.def * 4;
99     if(other_stats->vit > other->start.vit * 4) other_stats->vit =
100        ↪ other->start.vit * 4;
101 }
102
103 void player_turn(int cap) {
104     if(finished()) {
105         return;
106     }
107     Stats jc, js, ac, as;
108     jc = joueur.current;
109     ac = adversaire.current;
110     applyCapacite(joueur.capacites + cap, &joueur, &adversaire, &js,
111                   &as);
112     int i;
113     for(i = 0; i < 20; ++i) {
114         float t = i / 19.;
115         joueur.current = Stats_interpolate(jc, js, t);
116         adversaire.current = Stats_interpolate(ac, as, t);
117         affichage();
118         afficher_message(joueur.capacites[cap].message, &joueur,
119                           &adversaire);
120         SDL_Flip(ecran);
121         SDL_Delay(100);
122     }
123 }
124
125 void opponent_turn() {
126     if(finished()) {
127         return;
128     }
129     Stats jc, js, ac, as;
130     int other_cap = rand() % 2;
131     if(other_cap == 1) {
132         other_cap += rand() % 3;
133     }

```

```

128
129     jc = joueur.current;
130     ac = adversaire.current;
131     applyCapacite(adversaire.capacites + other_cap, &adversaire,
132                   &joueur, &as, &js);
133     int i;
134     for(i = 0; i < 20; ++i) {
135         float t = i / 19.;
136         joueur.current = Stats_interpolate(jc, js, t);
137         adversaire.current = Stats_interpolate(ac, as, t);
138         affichage();
139         afficher_message(adversaire.capacites[other_cap].message,
140                           &adversaire, &joueur);
141         SDL_Flip(ecran);
142         SDL_Delay(100);
143     }
144 }
145
146 void display_winner() {
147     char buffer[1024];
148     if(adversaire.current.vie <= 0) {
149         roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur /
150                         & 8, 3 * hauteur / 4, 15, 0, 53, 0, 204);
151         sprintf(buffer, "Victoire de %s", joueur.name);
152     } else {
153         roundedBoxRGBA(ecran, largeur / 8, hauteur / 4, 7 * largeur /
154                         & 8, 3 * hauteur / 4, 15, 53, 0, 0, 204);
155         sprintf(buffer, "Défaite de %s", joueur.name);
156     }
157     stringRGBA(ecran, 2 * largeur / 8, hauteur / 2, buffer, 255,
158                 & 255, 255, 255);
159 }
160
161 int finished() {
162     return joueur.current.vie <= 0 || adversaire.current.vie <= 0;
163 }
```

```

1  /* +-----+ */
2  /* / FICHIER : include/personnage.h / */
3  /* +-----+ */
```

```

4
5 #ifndef DEF_HEADER_ESGI_FIGHT_PERSONNAGE
6 #define DEF_HEADER_ESGI_FIGHT_PERSONNAGE
7
8 #include "stats.h"
9 #include "capacite.h"
10
11 typedef struct Personnage Personnage;
12 struct Personnage {
13     char name[50];
14     Stats start;
15     Stats current;
16     unsigned char face[1024];
17     unsigned char back[1024];
18     Capacite capacites[4];
19     int nb_caps;
20 };
21
22 Personnage Personnage_creer(const char * name, Stats stats);
23
24 int Personnage_load(Personnage * perso, const char * path);
25
26 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : src/personnage.c | */
3 /* +-----+ */
4
5 #include "../include/personnage.h"
6
7 #include <stdio.h>
8 #include <string.h>
9 #include "../include/draw.h"
10
11 Personnage Personnage_creer(const char * name, Stats stats) {
12     Personnage perso;
13     strcpy(perso.name, name);
14     perso.start = stats;
15     perso.current = stats;
16     return perso;

```

```
17 }
18
19 static int cap_load(FILE * fich, Personnage * perso) {
20     char buffer[50];
21     int i = 0;
22     for(;;) {
23         fscanf(fich, "%s", buffer);
24         if(strcmp(buffer, "}") == 0) {
25             break;
26         } else if(strcmp(buffer, "load") == 0) {
27             fscanf(fich, "%s", buffer);
28             if(! Capacite_load(buffer, perso->capacites + i)) {
29                 fprintf(stderr, "Erreur lecture capacite \"%s\"\n", buffer);
30             }
31             Capacite_debug(perso->capacites + i);
32             ++i;
33         }
34     }
35     perso->nb_caps = i;
36     return 1;
37 }
38
39 int Personnage_load(Personnage * perso, const char * path) {
40     FILE * fich = NULL;
41     if((fich = fopen(path, "r")) == NULL) {
42         fprintf(stderr, "Erreur lecture d'un personnage \"%s\"\n",
43             ↪ path);
44         return 0;
45     }
46     if(fscanf(fich, "name : %s\n", perso->name) != 1
47     || fscanf(fich, "vie : %d\n", &(perso->start.vie)) != 1
48     || fscanf(fich, "atk : %d\n", &(perso->start.atk)) != 1
49     || fscanf(fich, "def : %d\n", &(perso->start.def)) != 1
50     || fscanf(fich, "vit : %d\n", &(perso->start.vit)) != 1) {
51         fprintf(stderr, "Erreur de lecture des stats pour \"%s\"\n",
52             ↪ path);
53         goto error;
54     }
55     perso->current = perso->start;
56
57     unsigned char * draw = perso->face;
```

```

56     fscanf(fich, "face : {\n");
57
58     if(! data_load(fich, draw)) {
59         fprintf(stderr, "Erreur lecture dessin de face pour \"%s\"\n",
60                 → path);
61         goto error;
62     }
63
64     draw = perso->back;
65     fscanf(fich, "back : {\n");
66
67     if(! data_load(fich, draw)) {
68         fprintf(stderr, "Erreur lecture dessin de dos pour \"%s\"\n",
69                 → path);
70         goto error;
71     }
72
73     fscanf(fich, "capacities : {\n");
74
75     if(! cap_load(fich, perso)) {
76         fprintf(stderr, "Erreur lecture dessin de dos pour \"%s\"\n",
77                 → path);
78         goto error;
79     }
80
81     fclose(fich);
82
83     printf("\\"%s\\" chargé.\n", path);
84     return 1;
85
86     error :
87     fclose(fich);
88     return 0;
89 }
```

```

1  /* +-----+ */
2  /* / FICHIER : include/stats.h / */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_FIGHT_STATS
```

```
6 #define DEF_HEADER_ESGI_FIGHT_STATS
7
8 typedef struct Stats Stats;
9 struct Stats {
10     int vie;
11     int atk;
12     int def;
13     int vit;
14 };
15
16 typedef enum {
17     S_NONE = 0,
18     S_VALUE = 1,
19     S_VIE = 2,
20     S_ATK = 3,
21     S_DEF = 4,
22     S_VIT = 5
23 } Stat;
24
25 Stats Stats_interpolate(Stats first, Stats second, float t);
26
27 #endif
```

```
1 /* +-----+ */
2 /* | FICHIER : src/stats.c | */
3 /* +-----+ */
4
5 #include "../include/stats.h"
6
7 Stats Stats_interpolate(Stats first, Stats second, float t) {
8     return (Stats){
9         .vie = first.vie * (1 - t) + second.vie * t,
10        .atk = first.atk * (1 - t) + second.atk * t,
11        .def = first.def * (1 - t) + second.def * t,
12        .vit = first.vit * (1 - t) + second.vit * t
13    };
14 }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/draw.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_FIGHT_DRAW
6 #define DEF_HEADER_ESGI_FIGHT_DRAW
7
8 #include <SDL/SDL.h>
9
10 #include "personnage.h"
11
12 extern const int largeur;
13 extern const int hauteur;
14 extern const char * titre;
15 extern SDL_Surface * ecran;
16
17 typedef enum {
18     NOSHAPE    = 0,
19     S_LINE      = 1,
20     S_CIRCLE    = 2,
21     S_POLYGON   = 3
22 } Shape;
23
24 void barre_vie(int sx, int sy, int ex, int ey, int value, int
25 → max_value);
26
27 void barre_stat(int sx, int sy, int ex, int ey, int value, int
28 → start_value, float r, float g, float b);
29
30 void Personnage_afficher_stats(const Personnage * perso, int x,
31 → int y);
32
33 void draw_data(unsigned char * data, int cx, int cy, int s);
34
35 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : src/draw.c | */

```

```
3  /* +-----+ */
4
5 #include "../include/draw.h"
6
7 #include <stdio.h>
8 #include <SDL/SDL_gfxPrimitives.h>
9
10 const int largeur = 800;
11 const int hauteur = 600;
12 const char * titre = "ESGI fight";
13 SDL_Surface * ecran = NULL;
14
15 void barre_vie(int sx, int sy, int ex, int ey, int value, int
16    ↪ max_value) {
16    ↪ roundedBoxRGBA(ecran, sx, sy, ex, ey, 3, 51, 51, 51, 255);
17    int i;
18    for(i = 1; i < 4; ++i) {
19        vlineRGBA(ecran, sx + (i / 4.) * (ex - sx), sy, ey, 102, 102,
19        ↪ 102, 255);
20    }
21    if(value <= 0) {
22        return;
23    }
24    float t = (float)value / max_value;
25    float r, g, b;
26    if(t > 0.6) {
27        r = 0.; g = 1.; b = 0.;
28    } else if(t > 0.3) {
29        r = (0.6 - t) / 0.3; g = 1.; b = 0.;
30    } else {
31        r = 1.; g = t / 0.3; b = 0.;
32    }
33    roundedBoxRGBA(ecran, sx, sy, sx + t * (ex - sx), ey, 3, 51 +
33    ↪ 153 * r, 51 + 153 * g, 51 + 153 * b, 255);
34    if(t * (ex - sx) > 6) {
35        roundedBoxRGBA(ecran, sx + 3, sy + 3, sx + t * (ex - sx) - 3,
35        ↪ ey - 3, 2, 51 + 102 * r, 51 + 102 * g, 51 + 102 * b, 255);
36    }
37    char buffer[100];
38    sprintf(buffer, "%d / %d", value, max_value);
39    stringRGBA(ecran, sx + 15, sy + 2, buffer, 255, 255, 255, 255);
```

```

40 }
41
42 void barre_stat(int sx, int sy, int ex, int ey, int value, int
→ start_value, float r, float g, float b) {
43     roundedBoxRGB(ecran, sx, sy, ex, ey, 3, 102, 102, 102, 255);
44     int i;
45     for(i = 1; i < 6; ++i) {
46         vlineRGB(ecran, sx + (pow(2., i) / pow(2., 6.)) * (ex - sx),
→ sy, ey, 153, 153, 153, 255);
47     }
48     start_value *= 4;
49     if(value > start_value) {
50         value = start_value;
51     }
52     float t = (float)value / start_value;
53     roundedBoxRGB(ecran, sx, sy, sx + t * (ex - sx), ey, 3, 51 +
→ 153 * r, 51 + 153 * g, 51 + 153 * b, 255);
54     if(t * (ex - sx) > 6) {
55         roundedBoxRGB(ecran, sx + 3, sy + 3, sx + t * (ex - sx) - 3,
→ ey - 3, 2, 51 + 102 * r, 51 + 102 * g, 51 + 102 * b, 255);
56     }
57     char buffer[100];
58     sprintf(buffer, "%d", value);
59     stringRGB(ecran, sx + 15, sy + 2, buffer, 255, 255, 255, 255);
60 }
61
62 void Personnage_afficher_stats(const Personnage * perso, int x,
→ int y) {
63     int sx = x - largeur / 4;
64     int sy = y - hauteur / 9;
65     int ex = x + largeur / 4;
66     int ey = y + hauteur / 13;
67     roundedBoxRGB(ecran, sx, sy, ex, ey, 20, 255, 255, 255, 255);
68     roundedBoxRGB(ecran, sx + 5, sy + 5, ex - 5, ey - 5, 15, 204,
→ 204, 204, 255);
69     stringRGB(ecran, sx + 15, sy + 15, perso->name, 51, 51, 51,
→ 255);
70     hlineRGB(ecran, sx + 25, ex - 25, sy + 28, 51, 51, 51, 255);
71     stringRGB(ecran, sx + 15, sy + 35, "Vie : ", 51, 51, 51, 255);
72     barre_vie(sx + 60, sy + 32, ex - 15, sy + 44,
→ perso->current.vie, perso->start.vie);

```

```
73     stringRGB(ecran, sx + 15, sy + 50, "Attaque : ", 51, 51, 51,
    ↵ 255);
74     barre_stat(sx + 100, sy + 47, ex - 15, sy + 59,
    ↵ perso->current.atk, perso->start.atk, 1., 0., 0.);
75     stringRGB(ecran, sx + 15, sy + 65, "Defense : ", 51, 51, 51,
    ↵ 255);
76     barre_stat(sx + 100, sy + 62, ex - 15, sy + 74,
    ↵ perso->current.def, perso->start.def, 0., 0., 1.);
77     stringRGB(ecran, sx + 15, sy + 80, "Vitesse : ", 51, 51, 51,
    ↵ 255);
78     barre_stat(sx + 100, sy + 77, ex - 15, sy + 89,
    ↵ perso->current.vit, perso->start.vit, 1., 1., 0.);
79 }
80
81 void draw_data(unsigned char * data, int cx, int cy, int s) {
82     while(*data) {
83         Shape shape = *(data++);
84         switch(shape) {
85             case S_CIRCLE : {
86                 int x, y, d;
87                 int r, g, b, a;
88                 x = (char)(*(data++));
89                 y = (char)(*(data++));
90                 d = *(data++);
91                 r = *(data++);
92                 g = *(data++);
93                 b = *(data++);
94                 a = *(data++);
95                 x = cx + x * s / 100.;
96                 y = cy + y * s / 100.;
97                 d = d * s / 100.;
98                 filledCircleRGB(ecran, x, y, d, r, g, b, a);
99             } break;
100
101             case S_LINE : {
102                 int sx, sy, ex, ey, d;
103                 int r, g, b, a;
104                 sx = (char)(*(data++));
105                 sy = (char)(*(data++));
106                 ex = (char)(*(data++));
107                 ey = (char)(*(data++));
```

```

108     d = *(data++);
109     r = *(data++);
110     g = *(data++);
111     b = *(data++);
112     a = *(data++);
113     sx = cx + sx * s / 100.;
114     sy = cy + sy * s / 100.;
115     ex = cx + ex * s / 100.;
116     ey = cy + ey * s / 100.;
117     d = d * s / 100.;
118     thickLineRGBA(ecran, sx, sy, ex, ey, d, r, g, b, a);
119 } break;
120
121 case S_POLYGON : {
122     Sint16 xs[32];
123     Sint16 ys[32];
124     int n, i;
125     int r, g, b, a;
126     n = *(data++);
127     for(i = 0; i < n; ++i) {
128         xs[i] = (char)(*(data++));
129         ys[i] = (char)(*(data++));
130         xs[i] = cx + xs[i] * s / 100.;
131         ys[i] = cy + ys[i] * s / 100.;
132     }
133     r = *(data++);
134     g = *(data++);
135     b = *(data++);
136     a = *(data++);
137     filledPolygonRGBA(ecran, xs, ys, n, r, g, b, a);
138 } break;
139
140     default : break;
141 }
142 }
143 }
144
145 int data_load(FILE * fich, unsigned char * draw) {
146     char buffer[50];
147     for(;;) {
148         if(fscanf(fich, "%s", buffer) != 1) {

```

```
149     fprintf(stderr, "Erreur lecture commande\n");
150     return 0;
151 }
152 if(strcmp(buffer, "}") == 0) {
153     break;
154 } else if(strcmp(buffer, "circle") == 0) {
155     int x, y, d;
156     int r, g, b, a;
157     if(fscanf(fich, "%d, %d, %d rgba %d, %d, %d, %d\n", &x, &y,
158             &d, &r, &g, &b, &a) != 7) {
159         fprintf(stderr, "Erreur lecture cercle\n");
160         return 0;
161     }
162     *(draw++) = (unsigned char)S_CIRCLE;
163     *(draw++) = (unsigned char)x;
164     *(draw++) = (unsigned char)y;
165     *(draw++) = (unsigned char)d;
166     *(draw++) = (unsigned char)r;
167     *(draw++) = (unsigned char)g;
168     *(draw++) = (unsigned char)b;
169     *(draw++) = (unsigned char)a;
170 } else if(strcmp(buffer, "line") == 0) {
171     int sx, sy, ex, ey, d;
172     int r, g, b, a;
173     if(fscanf(fich, "%d, %d, %d, %d, %d rgba %d, %d, %d, %d\n",
174             &sx, &sy, &ex, &ey, &d, &r, &g, &b, &a) != 9) {
175         fprintf(stderr, "Erreur lecture line\n");
176         return 0;
177     }
178     *(draw++) = (unsigned char)S_LINE;
179     *(draw++) = (unsigned char)sx;
180     *(draw++) = (unsigned char)sy;
181     *(draw++) = (unsigned char)ex;
182     *(draw++) = (unsigned char)ey;
183     *(draw++) = (unsigned char)d;
184     *(draw++) = (unsigned char)r;
185     *(draw++) = (unsigned char)g;
186     *(draw++) = (unsigned char)b;
187     *(draw++) = (unsigned char)a;
188 } else if(strcmp(buffer, "polygon") == 0) {
189     Sint16 xs[32];
```

```

188     Sint16 ys[32];
189     int n, x, y, i;
190     int r, g, b, a;
191     if(fscanf(fich, "%d %d, %d", &n, &x, &y) != 3) {
192         fprintf(stderr, "Erreur lecture polygon : init\n");
193         return 0;
194     }
195     xs[0] = x;
196     ys[0] = y;
197     for(i = 1; i < n; ++i) {
198         if(fscanf(fich, ", %d, %d", &x, &y) != 2) {
199             fprintf(stderr, "Erreur lecture polygon : vertex %d\n",
200                     i);
201             return 0;
202         }
203         xs[i] = x;
204         ys[i] = y;
205     }
206     if(fscanf(fich, " rgba %d, %d, %d, %d\n", &r, &g, &b, &a) !=
207         4) {
208         fprintf(stderr, "Erreur lecture polygon : color\n");
209         return 0;
210     }
211     *(draw++) = (unsigned char)S_POLYGON;
212     *(draw++) = (unsigned char)n;
213     for(i = 0; i < n; ++i) {
214         *(draw++) = (unsigned char)(xs[i]);
215         *(draw++) = (unsigned char)(ys[i]);
216     }
217     *(draw++) = (unsigned char)r;
218     *(draw++) = (unsigned char)g;
219     *(draw++) = (unsigned char)b;
220     *(draw++) = (unsigned char)a;
221 }
222 *draw = (unsigned char)NOSHAPE;
223 return 1;
}

```

```
1  /* +-----+ */
2  /* | FICHIER : include/capacite.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_FIGHT_CAPACITE
6 #define DEF_HEADER_ESGI_FIGHT_CAPACITE
7
8 #include "action.h"
9
10 typedef struct Capacite Capacite;
11 struct Capacite {
12     char nom[64];
13     char message[1024];
14     Action * action;
15 };
16
17 int Capacite_update(Capacite * cap, const char * data);
18
19 int Capacite_load(const char * path, Capacite * cap);
20
21 void Capacite_debug(const Capacite * cap);
22
23 #endif
```

```
1  /* +-----+ */
2  /* | FICHIER : src/capacite.c | */
3  /* +-----+ */
4
5 #include "../include/capacite.h"
6
7 #include <stdio.h>
8 #include <string.h>
9
10 int Capacite_update(Capacite * cap, const char * data) {
11     char buffer[128];
12     if(strlen(data) < 3) {
13         return 1;
14     }
15     sscanf(data, "%s", buffer);
16     if(strcmp(buffer, "action") == 0) {
```

```

17     return Action_read(&(cap->action), data + strlen(buffer) + 1,
18     ↪ data + strlen(data));
19 } else if(strcmp(buffer, "message") == 0) {
20     strcpy(cap->message, data + strlen(buffer) + 1);
21 } else if(strcmp(buffer, "nom") == 0) {
22     strcpy(cap->nom, data + strlen(buffer) + 1);
23 }
24 return 1;
25 }
26
27 int Capacite_load(const char * path, Capacite * cap) {
28     FILE * file = NULL;
29     if((file = fopen(path, "r")) == NULL) {
30         return 0;
31     }
32     char buffer[1024];
33     int i;
34     strcpy(cap->message, "");
35     cap->action = NULL;
36     int car;
37     while((car = fgetc(file)) != EOF) {
38         i = 0;
39         do {
40             buffer[i++] = car;
41             car = fgetc(file);
42         } while(car != EOF && car != '\n');
43         buffer[i] = '\0';
44         Capacite_update(cap, buffer);
45     }
46     fclose(file);
47     return 1;
48 }
49
50 void Capacite_debug(const Capacite * cap) {
51     printf("Action : {\n");
52     printf(" - nom : %s\n", cap->nom);
53     printf(" - message : %s\n", cap->message);
54     Action * action;
55     for(action = cap->action; action != NULL; action = action->next)
      ↪ {
        printf(" - action : "); Action_debug(action); printf("\n");

```

```
56     }
57     printf("%}\n");
58 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/action.h / */
3  /* +-----+ */

4
5  #ifndef DEF_HEADER_ESGI_FIGHT_ACTION
6  #define DEF_HEADER_ESGI_FIGHT_ACTION

7
8  #include "stats.h"

9
10 typedef enum {
11     T_NONE = 0,
12     T_SELF = 1,
13     T_OTHER = 2
14 } Target;

15
16 typedef enum {
17     O_NONE = 0,
18     O_EVAL = 1,
19     O_AFFECT = 2,
20     O_ADD = 3,
21     O_AFF_ADD = 4,
22     O_SUB = 5,
23     O_AFF_SUB = 6,
24     O_MUL = 7,
25     O_AFF_MUL = 8,
26     O_DIV = 9,
27     O_AFF_DIV = 10
28 } Operation;

29
30 typedef struct Action Action;
31 struct Action {
32     Stat stat;
33     Target target;
34     float value;
35     Operation op;
36     Action * left;
```

```

37     Action * right;
38     Action * next;
39 }
40
41 Target Target_trad(const char * v);
42
43 Stat Stat_trad(const char * v);
44
45 Action * Action_alloc(Action * next);
46
47 int Action_read(Action ** action, const char * start, const char *
→   end);
48
49 void Action_debug(const Action * action);
50
51 float Action_eval(const Action * action, Stats * self_stats, Stats
→   * other_stats);
52
53 #endif

```

```

1 /* +-----+ */
2 /* / FICHIER : src/action.c / */
3 /* +-----+ */
4
5 #include "../include/action.h"
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <string.h>
10
11 static int isop(char c) {
12     return c == '+'
13     || c == '-'
14     || c == '*'
15     || c == '/';
16 }
17
18 static Operation getop(char c) {
19     switch(c) {
20         case '=' : return O_AFFECT;

```

```
21     case '+' : return O_ADD;
22     case '-' : return O_SUB;
23     case '*' : return O_MUL;
24     case '/' : return O_DIV;
25     default : return O_NONE;
26   }
27 }
28
29 static Operation getaffop(char c) {
30   Operation op = getop(c);
31   if(op > O_AFFECT) {
32     return op + 1;
33   }
34   return O_AFFECT;
35 }
36
37 static int getprio(char c) {
38   switch(c) {
39     case '=' : return 3;
40     case '+' : return 4;
41     case '-' : return 4;
42     case '*' : return 5;
43     case '/' : return 5;
44     default : return 0;
45   }
46 }
47
48 Target Target_trad(const char * v) {
49   if(strcmp(v, "self") == 0) {
50     return T_SELF;
51   } else if(strcmp(v, "other") == 0) {
52     return T_OTHER;
53   }
54   return T_NONE;
55 }
56
57 Stat Stat_trad(const char * v) {
58   if(strcmp(v, "vie") == 0) {
59     return S_VIE;
60   } else if(strcmp(v, "atk") == 0) {
61     return S_ATK;
```

```

62 } else if(strcmp(v, "def") == 0) {
63     return S_DEF;
64 } else if(strcmp(v, "vit") == 0) {
65     return S_VIT;
66 }
67 return S_NONE;
68 }

69 Action * Action_alloc(Action * next) {
70     Action * action = NULL;
71     if((action = (Action *)malloc(sizeof(Action))) == NULL) {
72         return NULL;
73     }
74     action->stat = S_VALUE;
75     action->target = T_NONE;
76     action->op = O_EVAL;
77     action->value = 0.f;
78     action->left = NULL;
79     action->right = NULL;
80     action->next = next;
81     return action;
82 }
83

84 static int Action_read_eval(Action ** action, const char * start,
85 → const char * end) {
86     const char * c;
87     char target[50];
88     char stat[50];
89     if((*action = Action_alloc(*action)) == NULL) {
90         fprintf(stderr, "Erreur allocation action.\n");
91         exit(EXIT_FAILURE);
92     }
93     for(c = start; c != end; ++c) {
94         if(*c == '$') {
95             sscanf(c + 1, "%[^.].%s", target, stat);
96             (*action)->stat = Stat_trad(stat);
97             (*action)->target = Target_trad(target);
98             return 1;
99         }
100    }
101    float val;

```

```
102     if(sscanf(start, "%f", &val) != 1) {
103         fprintf(stderr, "Erreur evaluation de \"%s\".", start);
104         free(*action);
105         *action = NULL;
106         return 0;
107     }
108     (*action)->stat = S_VALUE;
109     (*action)->value = val;
110     return 1;
111 }
112
113 int Action_read(Action ** action, const char * start, const char *
114 → end) {
114     int par = 0;
115     const char * op_offset = NULL;
116     int op_prio = -1;
117     const char * c;
118     int has_content = 0;
119     for(c = start; c != end; ++c) {
120         if(*c == '(') {
121             ++par;
122             continue;
123         }
124         if(*c == ')') {
125             --par;
126             continue;
127         }
128         if(*c == ' ' || *c == '\t' || *c == '\n') {
129             continue;
130         }
131         if(par) {
132             continue;
133         }
134         has_content = 1;
135         if(isop(*c) || *c == '=') {
136             int prio = getprio(*c);
137             if(op_prio <= 0 || prio < op_prio) {
138                 op_offset = c;
139                 op_prio = prio;
140             }
141             continue;
142 }
```

```

142     }
143 }
144 if(! has_content) {
145     for(; *start != '(' && start < end; ++start);
146     for(; *end != ')' && start < end; --end);
147     if(start >= end) {
148         return 0;
149     }
150     ++start;
151     ++end;
152     return Action_read(action, start, end);
153 }
154 if(op_prio <= 0) {
155     return Action_read_eval(action, start, end);
156 }
157 if((*action = Action_alloc(*action)) == NULL) {
158     fprintf(stderr, "Erreur allocation action.\n");
159     exit(EXIT_FAILURE);
160 }
161 if(*op_offset == '=') {
162     (*action)->op = getaffop(*op_offset - 1));
163     if(isop(*op_offset - 1))) {
164         return Action_read(&((*action)->left), start, op_offset - 1)
165         && Action_read(&((*action)->right), op_offset + 1, end);
166     }
167     return Action_read(&((*action)->left), start, op_offset)
168     && Action_read(&((*action)->right), op_offset + 1, end);
169 }
170 (*action)->op = getop(*op_offset);
171 return Action_read(&((*action)->left), start, op_offset)
172 && Action_read(&((*action)->right), op_offset + 1, end);
173 }

174 void Action_debug(const Action * action) {
175     if(! action) {
176         return;
177     }
178     if(action->op == O_EVAL) {
179         if(action->stat == S_VALUE) {
180             printf("%g", action->value);
181             return;
182         }

```

```
183     }
184     switch(action->target) {
185         case T_SELF : printf("self"); break;
186         case T_OTHER : printf("other"); break;
187         default : break;
188     }
189     printf(".");
190     switch(action->stat) {
191         case S_VIE : printf("vie"); break;
192         case S_ATK : printf("atk"); break;
193         case S_DEF : printf("def"); break;
194         case S_VIT : printf("vit"); break;
195         default : break;
196     }
197     return;
198 }
199 printf("(");
200 Action_debug(action->left);
201 switch(action->op) {
202     case O_AFFECT : printf(" = "); break;
203     case O_ADD : printf(" + "); break;
204     case O_SUB : printf(" - "); break;
205     case O_MUL : printf(" * "); break;
206     case O_DIV : printf(" / "); break;
207     case O_AFF_ADD : printf(" += "); break;
208     case O_AFF_SUB : printf(" -= "); break;
209     case O_AFF_MUL : printf(" *= "); break;
210     case O_AFF_DIV : printf(" /= "); break;
211     default : break;
212 }
213 Action_debug(action->right);
214 printf(")");
215 }

216 float Action_eval(const Action * action, Stats * self_stats, Stats
217 ← * other_stats) {
218     if(! action) {
219         return 0.f;
220     }
221     if(action->op == O_EVAL) {
222         if(action->stat == S_VALUE) {
```

```

223     return action->value;
224 }
225 Stats * look = NULL;
226 switch(action->target) {
227     case T_SELF : look = self_stats; break;
228     case T_OTHER : look = other_stats; break;
229     default : return 0.f;
230 }
231 switch(action->stat) {
232     case S_VIE : return look->vie;
233     case S_ATK : return look->atk;
234     case S_DEF : return look->def;
235     case S_VIT : return look->vit;
236     default : break;
237 }
238 return 0.f;
239 }
240 if(action->op == O_AFFECT
241 || action->op == O_AFF_ADD
242 || action->op == O_AFF_SUB
243 || action->op == O_AFF_MUL
244 || action->op == O_AFF_DIV) {
245     int * left = NULL;
246     Action * leftNode = action->left;
247     Stats * look = NULL;
248     switch(leftNode->target) {
249         case T_SELF : look = self_stats; break;
250         case T_OTHER : look = other_stats; break;
251         default : return 0.f;
252     }
253     switch(leftNode->stat) {
254         case S_VIE : left = &(look->vie); break;
255         case S_ATK : left = &(look->atk); break;
256         case S_DEF : left = &(look->def); break;
257         case S_VIT : left = &(look->vit); break;
258         default : return 0.f;
259     }
260     float right = Action_eval(action->right, self_stats,
261     → other_stats);
262     switch(action->op) {
263         case O_AFFECT : return (*left) = right;

```

```
263     case O_AFF_ADD : return (*left) += right;
264     case O_AFF_SUB : return (*left) -= right;
265     case O_AFF_MUL : return (*left) *= right;
266     case O_AFF_DIV : return (*left) /= right;
267     default : return 0.f;
268 }
269 } else {
270     float left = Action_eval(action->left, self_stats,
271     ↪ other_stats);
272     float right = Action_eval(action->right, self_stats,
273     ↪ other_stats);
274     switch(action->op) {
275         case O_ADD : return left + right;
276         case O_SUB : return left - right;
277         case O_MUL : return left * right;
278         case O_DIV : return left / right;
279         default : return 0.f;
280     }
}
```

A.12 Programmation modulaire

A.12.1 Entraînement

Correction 115 (★★ Mise en place d'un Makefile).

```

1 CC= gcc
2 CFLAGS= -O2 -Wall -Wextra -Werror -ansi
3 CLIBS= -lm
4 EXE= executable
5 OBJ= main.o \
6     point.o \
7     triangle.o
8
9 $(EXE) : $(OBJ)
10    $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
11
12 main.o : main.c triangle.h
13 point.o : point.c point.h
14 triangle.o : triangle.c triangle.h point.h
15
16 %.o : %.c
17    $(CC) $(CFLAGS) -c $<
18
19 clean :
20    rm -rf $(OBJ)
21    rm -rf $(EXE)
```

```

1 /* +-----+ */
2 /* / FICHIER : main.c / */
3 /* +-----+ */
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "triangle.h"
8
9 int main() {
10     Triangle triangle = Triangle_creer(
11         Point_creer(0, 0),
12         Point_creer(4, 0),
```

```
13     Point_creer(0, 3)
14 );
15 Triangle_afficher(&triangle);
16 printf("\nPerimetre : %g\n", Triangle_perimetre(&triangle));
17 exit(EXIT_SUCCESS);
18 }
```

```
1 /* +-----+ */
2 /* / FICHIER : point.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_POINT
5 #define DEF_HEADER_POINT
6
7 typedef struct Point Point;
8 struct Point {
9     double x;
10    double y;
11 };
12
13 Point Point_creer(double x, double y);
14
15 void Point_afficher(const Point * p);
16
17 double Point_distance(const Point * first, const Point * second);
18
19 #endif
```

```
1 /* +-----+ */
2 /* / FICHIER : point.c / */
3 /* +-----+ */
4 #include "point.h"
5
6 #include <stdio.h>
7 #include <math.h>
8
9 Point Point_creer(double x, double y) {
10    Point p = {x, y};
11    return p;
12 }
```

```

14 void Point_afficher(const Point * p) {
15     printf("(%.g, %.g)", p->x, p->y);
16 }
17
18 double Point_distance(const Point * first, const Point * second) {
19     return sqrt(pow(second->x - first->x, 2) + pow(second->y -
20                 first->y, 2));
}

```

```

1  /* +-----+ */
2  /* | FICHIER : triangle.h | */
3  /* +-----+ */
4 #ifndef DEF_HEADER_TRIANGLE
5 #define DEF_HEADER_TRIANGLE
6
7 #include "point.h"
8
9 typedef struct Triangle Triangle;
10 struct Triangle {
11     Point first;
12     Point second;
13     Point third;
14 };
15
16 Triangle Triangle_creer(Point first, Point second, Point third);
17
18 void Triangle_afficher(const Triangle * triangle);
19
20 double Triangle_perimetre(const Triangle * triangle);
21
22 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : triangle.c | */
3  /* +-----+ */
4 #include "triangle.h"
5
6 #include <stdio.h>
7
8 Triangle Triangle_creer(Point first, Point second, Point third) {

```

```
9     Triangle triangle = {first, second, third};  
10    return triangle;  
11 }  
12  
13 void Triangle_afficher(const Triangle * triangle) {  
14     printf("{Triangle : ");  
15     Point_afficher(&(triangle->first));  
16     printf(", ");  
17     Point_afficher(&(triangle->second));  
18     printf(", ");  
19     Point_afficher(&(triangle->third));  
20     printf("}");  
21 }  
22  
23 double Triangle_perimetre(const Triangle * triangle) {  
24     return  
25         Point_distance(&(triangle->first), &(triangle->second))  
26         + Point_distance(&(triangle->second), &(triangle->third))  
27         + Point_distance(&(triangle->third), &(triangle->first));  
28 }
```

Correction 116 (★★★ Profilage d'un code).

```
1 /* +-----+ */  
2 /* / FICHIER : main.c / */  
3 /* +-----+ */  
4 #include <stdio.h>  
5 #include <stdlib.h>  
6  
7 #define DO_PROFILE  
8 #include "profile.h"  
9  
10 START_FUNCTION(int, f, int x)  
11 END_FUNCTION(x * x)  
12  
13 START_FUNCTION(int, main)  
14     int a = 42;  
15     a = f(a);  
16     printf("a = %d\n", a);  
17     exit(EXIT_SUCCESS);
```

```
18 END_FUNCTION(0)
```

```

1  /* +-----+ */
2  /* | FICHIER : profile.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_PROFILE
5  #define DEF_HEADER_PROFILE

6
7  #include <stdio.h>
8
9  #define log(...) fprintf(stderr, __VA_ARGS__);
10
11 #ifdef DO_PROFILE
12
13 #define exit(V) log("> Exit in function %s at line %d with value
14    → %s\n", __FUNCTION__, __LINE__, #V); exit(V);
15 #define START_FUNCTION(type, name, ...) type name(__VA_ARGS__)
16    → \
17        log("# Definition of %s function %s(%s) in file \"%s\" at
18        → line %d\n", #type, #name, #__VA_ARGS__, __FILE__, __LINE__); \
19        log("< Starting function %s :\n", __FUNCTION__);
20 #define END_FUNCTION(V) log("> Ending function %s with return
21    → %s\n", __FUNCTION__, #V); return V; }

22
23 #else
24
25 #endif
26 #endif

```

Correction 117 (★★★ Implémentation table de hachage).

```

1 CC= gcc
2 CFLAGS= -O2 -Wall -Wextra -Werror -ansi
3 CLIBS= -lm
4 DEFS= -DBENCHMARKS

```

```
5  EXE= executable
6  OBJ= main.o \
7      arraylist.o \
8      hashmap.o
9
10 $(EXE) : $(OBJ)
11     $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
12
13 main.o : main.c arraylist.h hashmap.h
14 arraylist.o : arraylist.c arraylist.h
15 hashmap.o : hashmap.c hashmap.h
16
17 %.o : %.c
18     $(CC) $(CFLAGS) -c $< $(DEFS)
19
20 clean :
21     rm -rf $(OBJ)
22     rm -rf $(EXE)
```

```
1  /* +-----+ */
2  /* / FICHIER : main.c / */
3  /* +-----+ */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  #include "arraylist.h"
9  #include "hashmap.h"
10
11 #ifdef BENCHMARKS
12 #define CAPACITY    10000
13 #define ELEMENTS   1000000
14 #define RESEARCHES 10000
15 #else
16 #define CAPACITY 5
17 #define ELEMENTS 20
18#endif
19
20 int main() {
21     HashMap * map = HashMap_creer(CAPACITY);
```

```

22 ArrayList * liste = ArrayList_creer(CAPACITY);
23 long seed = time(NULL);
24 int value;
25 long i;
26 #ifdef BENCHMARKS
27     int count;
28     fprintf(stderr, "Filling ArrayList\n");
29     clock_t start, stop;
30     start = clock();
31 #endif
32     srand(seed);
33     for(i = 0; i < ELEMENTS; ++i) {
34         value = rand() % ELEMENTS;
35         ArrayList_ajouter(liste, value);
36     }
37 #ifdef BENCHMARKS
38     stop = clock();
39     fprintf(stderr, "Filling completed in %g s\n",
40             (double)(stop - start) / CLOCKS_PER_SEC);
41     fprintf(stderr, "Filling HashMap\n");
42     start = clock();
43 #endif
44     srand(seed);
45     for(i = 0; i < ELEMENTS; ++i) {
46         value = rand() % ELEMENTS;
47         HashMap_ajouter(map, value);
48     }
49 #ifdef BENCHMARKS
50     stop = clock();
51     fprintf(stderr, "Filling completed in %g s\n",
52             (double)(stop - start) / CLOCKS_PER_SEC);
53 #endif
54 #ifndef BENCHMARKS
55     HashMap_afficher(stdout, map);
56     ArrayList_afficher(stdout, liste);
57 #else
58     seed *= 42;
59     fprintf(stderr, "Research in ArrayList\n");
60     start = clock();
61     srand(seed);
62     count = 0;

```

```
63     for(i = 0; i < RESEARCHES; ++i) {
64         value = rand() % ELEMENTS;
65         count += ArrayList_compter(liste, value);
66     }
67     stop = clock();
68     fprintf(stderr, "Research completed in %g s\n",
69             (double)(stop - start) / CLOCKS_PER_SEC);
70     fprintf(stderr, "Counting %d elements\n", count);

71     fprintf(stderr, "Research in HashMap\n");
72     start = clock();
73     srand(seed);
74     count = 0;
75     for(i = 0; i < RESEARCHES; ++i) {
76         value = rand() % ELEMENTS;
77         count += HashMap_compter(map, value);
78     }
79     stop = clock();
80     fprintf(stderr, "Research completed in %g s\n",
81             (double)(stop - start) / CLOCKS_PER_SEC);
82     fprintf(stderr, "Counting %d elements\n", count);
83 #endif
84     HashMap_free(&map);
85     ArrayList_free(&liste);
86     exit(EXIT_SUCCESS);
87 }
88 }
```

```
1  /* +-----+ */
2  /* | FICHIER : arraylist.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_LISTE
5  #define DEF_HEADER_LISTE
6
7  #include <stdio.h>
8
9  typedef struct ArrayList ArrayList;
10 struct ArrayList;
11
12 ArrayList * ArrayList_creer(int capacite);
13 }
```

```

14 void ArrayList_free(ArrayList ** arraylist);
15
16 int ArrayList_ajouter(ArrayList * liste, int valeur);
17
18 void ArrayList_afficher(FILE * flow, const ArrayList * liste);
19
20 int ArrayList_compter(const ArrayList * liste, int valeur);
21
22 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : arraylist.c | */
3  /* +-----+ */
4 #include "arraylist.h"
5
6 #include <stdlib.h>
7
8 struct ArrayList {
9     int * data;
10    int taille;
11    int capacite;
12};
13
14 ArrayList * ArrayList_creer(int capacite) {
15    ArrayList * res = NULL;
16    if((res = (ArrayList *)malloc(sizeof(ArrayList))) == NULL) {
17        return NULL;
18    }
19    if(capacite > 0) {
20        if((res->data = (int *)malloc(capacite * sizeof(int)))
21           == NULL) {
22            free(res);
23            return NULL;
24        }
25    }
26    res->capacite = capacite;
27    res->taille = 0;
28    return res;
29}
30

```

```
31 void ArrayList_free(ArrayList ** arraylist) {
32     if(arraylist == NULL || *arraylist == NULL) {
33         return;
34     }
35     if((*arraylist)->data != NULL) {
36         free((*arraylist)->data);
37     }
38     free(*arraylist);
39     *arraylist = NULL;
40 }
41
42 static int ArrayList_update_capacity(ArrayList * liste) {
43     if(liste->taille < liste->capacite) {
44         return 1;
45     }
46     int * tmp = NULL;
47     int newCap = liste->capacite * 2 + 10;
48     if((tmp = (int *)realloc(liste->data, newCap * sizeof(int))) ==
49         NULL) {
50         return 0;
51     }
52     liste->data = tmp;
53     liste->capacite = newCap;
54     return 1;
55 }
56
57 int ArrayList_ajouter(ArrayList * liste, int valeur) {
58     if(! ArrayList_update_capacity(liste)) {
59         return 0;
60     }
61     liste->data[liste->taille] = valeur;
62     ++(liste->taille);
63     return 1;
64 }
65
66 void ArrayList_afficher(FILE * flow, const ArrayList * liste) {
67     int i;
68     fprintf(flow, "ArrayList<int> : [");
69     for(i = 0; i < liste->taille; ++i) {
70         if(i) fprintf(flow, ", ");
71         fprintf(flow, "%d", liste->data[i]);
```

```

71 }
72 fprintf(flow, "]\n");
73 }
74
75 int ArrayList_compter(const ArrayList * liste, int valeur) {
76     int count = 0;
77     int i;
78     for(i = 0; i < liste->taille; ++i) {
79         if(liste->data[i] == valeur)
80             ++count;
81     }
82     return count;
83 }
```

```

1  /* +-----+ */
2  /* | FICHIER : hashmap.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_HASHMAP
5  #define DEF_HEADER_HASHMAP
6
7  #include <stdio.h>
8
9  typedef struct HashMap HashMap;
10 struct HashMap;
11
12 HashMap * HashMap_creer(int capacite);
13
14 void HashMap_free(HashMap ** hashmap);
15
16 int HashMap_ajouter(HashMap * hashmap, int valeur);
17
18 void HashMap_afficher(FILE * flow, const HashMap * hashmap);
19
20 int HashMap_compter(const HashMap * hashmap, int valeur);
21
22 #endif
```

```

1  /* +-----+ */
2  /* | FICHIER : hashmap.c | */
3  /* +-----+ */
```

```
4 #include "hashmap.h"
5
6 #include <stdlib.h>
7
8 typedef struct HashMapElement HME;
9 struct HashMapElement {
10     int key;
11     int value;
12     HME * next;
13 };
14
15 struct HashMap {
16     HME ** data;
17     int capacite;
18 };
19
20 static HME * HME_alloc(int key, int value, HME * next);
21
22 static HME * HME_rechercher(HME * root, int key);
23
24 static void HME_free(HME ** node);
25
26 static HME * HME_alloc(int key, int value, HME * next) {
27     HME * res = NULL;
28     if((res = (HME *)malloc(sizeof(HME))) == NULL) {
29         return NULL;
30     }
31     res->key = key;
32     res->value = value;
33     res->next = next;
34     return res;
35 }
36
37 static HME * HME_rechercher(HME * root, int key) {
38     for(; root != NULL; root = root->next) {
39         if(root->key == key) {
40             return root;
41         }
42     }
43     return NULL;
44 }
```

```

45
46 static void HME_free(HME ** node) {
47     if(node == NULL || *node == NULL) {
48         return;
49     }
50     HME_free(&(*node)->next);
51     free(*node);
52     *node = NULL;
53 }
54
55 /* HashMap section */
56
57 HashMap * HashMap_creer(int capacite) {
58     HashMap * res = NULL;
59     if((res = (HashMap *)malloc(sizeof(HashMap))) == NULL) {
60         return NULL;
61     }
62     if((res->data = (HME **)calloc(capacite, sizeof(HME *))) ==
63         NULL) {
64         free(res);
65         return NULL;
66     }
67     res->capacite = capacite;
68     return res;
69 }
70
71 void HashMap_free(HashMap ** hashmap) {
72     if(hashmap == NULL || *hashmap == NULL) {
73         return;
74     }
75     int i;
76     for(i = 0; i < (*hashmap)->capacite; ++i) {
77         HME_free((*hashmap)->data + i);
78     }
79     free((*hashmap)->data);
80     free(*hashmap);
81     *hashmap = NULL;
82 }
83
84 static int hashInt(int valeur, int maxi) {
85     return valeur % maxi;

```

```
85 }
86
87 int HashMap_ajouter(HashMap * hashmap, int valeur) {
88     HME * current = NULL;
89     int hid = hashInt(valeur, hashmap->capacite);
90     if((current = HME_rechercher(*(hashmap->data + hid), valeur)) ==
91         → NULL) {
92         if((current = HME_alloc(valeur, 1, *(hashmap->data + hid))) ==
93             → NULL) {
94             return 0;
95         }
96         *(hashmap->data + hid) = current;
97     } else {
98         ++(current->value);
99     }
100    return 1;
101}
102
103 void HashMap_afficher(FILE * flow, const HashMap * hashmap) {
104     int i;
105     HME * current = NULL;
106     int count = 0;
107     fprintf(flow, "HashMap<int, int> : {");
108     for(i = 0; i < hashmap->capacite; ++i) {
109         for(current = *(hashmap->data + i); current != NULL; current =
110             → current->next) {
111             if(count++) fprintf(flow, ", ");
112             fprintf(flow, "%d : %d", current->key, current->value);
113         }
114         fprintf(flow, "}\n");
115     }
116     int HashMap_compter(const HashMap * hashmap, int valeur) {
117         int hid = hashInt(valeur, hashmap->capacite);
118         HME * current = HME_rechercher(*(hashmap->data + hid), valeur);
119         return (current) ? (current->value) : 0;
120     }
```

Correction 118 (★★★ Représentation d'un lexique).

```

1 CC= gcc
2 CFLAGS= -O2 -Wall -Wextra -Werror -ansi
3 CLIBS= -lm
4 DEFS= -DTEST
5 EXE= executable
6 OBJ= main.o \
    tree.o
8
9 $(EXE) : $(OBJ)
10   $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
11
12 main.o : main.c tree.h
13 tree.o : tree.c tree.h
14
15 %.o : %.c
16   $(CC) $(CFLAGS) -c $< $(DEFS)
17
18 clean :
19   rm -rf $(OBJ)
20   rm -rf $(EXE)

```

```

1 /* +-----+ */
2 /* / FICHIER : main.c / */
3 /* +-----+ */
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "tree.h"
8
9 #ifndef TEST
10 int main(int argc, char * argv[]) {
11
12     char * path = NULL;
13     if(argc > 1) {
14         path = argv[1];
15     }
16     if(path == NULL) {
17         printf("Attente : %s [FICHIER]\n", argv[0]);
18         exit(EXIT_FAILURE);
19     }

```

```
20 Node * root = NULL;
21 FILE * input = NULL;
22 if((input = fopen(path, "r")) == NULL) {
23     fprintf(stderr, "Erreur d'ouverture de \"%s\"\n", path);
24     exit(EXIT_FAILURE);
25 }
26 char buffer[100];
27 while(fscanf(input, "%s", buffer) == 1) {
28     Node_ajouter_chaine(&root, buffer);
29 }
30 Node_afficher(stdout, root);
31 Node_afficher_all(stderr, root);
32 Node_free(&root);
33 #else
34 int main() {
35     Node * root = NULL;
36     Node_ajouter_chaine(&root, "Test");
37     Node_ajouter_chaine(&root, "Preference");
38     Node_ajouter_chaine(&root, "Prefixe");
39     Node_ajouter_chaine(&root, "Test");
40     Node_ajouter_chaine(&root, "Alpha");
41     Node_ajouter_chaine(&root, "Alphabet");
42     Node_afficher(stdout, root);
43
44     Node_afficher_all(stderr, root);
45     Node_free(&root);
46 #endif
47     exit(EXIT_SUCCESS);
48 }
```

```
1 /* +-----+ */
2 /* / FICHIER : tree.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_TREE
5 #define DEF_HEADER_TREE
6
7 #include <stdio.h>
8
9 typedef struct Node Node;
10 struct Node {
```

```

11   char key;
12   int value;
13   Node * lower;
14   Node * upper;
15   Node * next;
16   Node * prev;
17 };
18
19 Node * Node_alloc(char key, int value);
20
21 void Node_free(Node ** node);
22
23 int Node_ajouter_chaine(Node ** node, const char * chaine);
24
25 void Node_afficher(FILE * flow, const Node * node);
26
27 void Node_afficher_all(FILE * flow, const Node * node);
28
29 #endif

```

```

1  /* +-----+ */
2  /* / FICHIER : tree.c / */
3  /* +-----+ */
4  #include "tree.h"
5
6  #include <stdlib.h>
7
8  Node * Node_alloc(char key, int value) {
9      Node * res = NULL;
10     if((res = (Node *)malloc(sizeof(Node))) == NULL) {
11         return NULL;
12     }
13     res->key = key;
14     res->value = value;
15     res->lower = NULL;
16     res->upper = NULL;
17     res->next = NULL;
18     res->prev = NULL;
19     return res;
20 }

```

```
21
22 void Node_free(Node ** node) {
23     if(node == NULL || *node == NULL) {
24         return;
25     }
26     Node_free(&(*node)->lower);
27     Node_free(&(*node)->upper);
28     Node_free(&(*node)->next);
29     free(*node);
30     *node = NULL;
31 }
32
33 int Node_ajouter_chaine(Node ** node, const char * chaine) {
34     Node * prev = *node;
35     for(;;) {
36         while(*node != NULL) {
37             if(*chaine < (*node)->key) {
38                 node = &(*node)->lower;
39             } else if(*chaine > (*node)->key) {
40                 node = &(*node)->upper;
41             } else {
42                 break;
43             }
44         }
45         if(*node == NULL) {
46             if((*node = Node_alloc(*chaine, 0)) == NULL) {
47                 return 0;
48             }
49             (*node)->prev = prev;
50         }
51         ++((*node)->value);
52         if(*chaine == '\0') {
53             break;
54         }
55         ++chaine;
56         node = &(*node)->next;
57         prev = *node;
58     }
59     return 1;
60 }
```

```

62 static void Node_afficher_aux(FILE * flow, const Node * node, char
63   * chaine, int offset) {
64   if(node == NULL) return;
65   Node_afficher_aux(flow, node->lower, chaine, offset);
66   if(*(chaine + offset) == '\0') {
67     fprintf(flow, "%s : %d\n", chaine, node->value);
68   }
69   Node_afficher_aux(flow, node->next, chaine, offset + 1);
70   Node_afficher_aux(flow, node->upper, chaine, offset);
71 }
72
73 void Node_afficher(FILE * flow, const Node * node) {
74   char buff[100] = "";
75   Node_afficher_aux(flow, node, buff, 0);
76 }
77
78 static void Node_afficher_all_aux(FILE * flow, const Node * node,
79   char * chaine, int offset) {
80   if(node == NULL) return;
81   Node_afficher_all_aux(flow, node->lower, chaine, offset);
82   if(node->key != '\0') {
83     *(chaine + offset) = node->key;
84     *(chaine + offset + 1) = '\0';
85     fprintf(flow, "%d : %s\n", node->value, chaine);
86   }
87   Node_afficher_all_aux(flow, node->next, chaine, offset + 1);
88   Node_afficher_all_aux(flow, node->upper, chaine, offset);
89 }
90
91 void Node_afficher_all(FILE * flow, const Node * node) {
92   char buff[100] = "";
93   Node_afficher_all_aux(flow, node, buff, 0);
94 }
```

A.13 Types génériques

A.13.1 Entraînement

Correction 119 (★★★ Trier une liste de points).

```
1 CC= gcc
2 CFLAGS= -O2 -Wall -Wextra -Werror -ansi
3 CLIBS= -lm
4 DEFS=
5 EXE= executable
6 OBJ= main.o \
    point.o
8
9 $(EXE) : $(OBJ)
10      $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
11
12 main.o : main.c point.h
13 point.o : point.c point.h
14
15 %.o : %.c
16      $(CC) $(CFLAGS) -c $< $(DEFS)
17
18 clean :
19      rm -rf $(OBJ)
20      rm -rf $(EXE)
```

```
1 /* +-----+ */
2 /* / FICHIER : main.c / */
3 /* +-----+ */
4 #define _GNU_SOURCE
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <math.h>
9
10 #include "point.h"
11
12 double Point_distanceCarre_origine(const Point * point) {
13     return point->x * point->x + point->y * point->y;
```

```

14 }
15
16 int Point_cmpDistanceOrigine(const Point * first, const Point *
17   ↪ second) {
17   double diff = Point_distanceCarre_origine(first) -
18     ↪ Point_distanceCarre_origine(second);
18   if(-1.e-9 < diff && diff < 1.e-9) {
19     return 0;
20   }
21   return (diff < 0) ? - 1 : 1;
22 }
23
24 int Point_cmpDistanceTarget(const Point * first, const Point *
25   ↪ second, Point * target) {
25   double diff = Point_distance(first, target) -
26     ↪ Point_distance(second, target);
26   if(-1.e-9 < diff && diff < 1.e-9) {
27     return 0;
28   }
29   return (diff < 0) ? (diff - 1) : (diff + 1);
30 }
31
32 int main() {
33   Point points[5] = {
34     Point_creer(-3, 0),
35     Point_creer(4, 0),
36     Point_creer(0, 3),
37     Point_creer(0, 1),
38     Point_creer(2, 3)
39   };
40   qsort(points, 5, sizeof(Point),
41     (int (*)(const void *, const void
42       ↪ *))&Point_cmpDistanceOrigine);
43   int i;
44   printf("Distance origine : \n");
45   for(i = 0; i < 5; ++i) {
46     Point_afficher(points + i);
47     printf(" %g\n", sqrt(Point_distanceCarre_origine(points +
48       ↪ i))));
48   }
Point target = Point_creer(3, 3);

```

```
49     qsort_r(points, 5, sizeof(Point),
50             (int (*)(const void *, const void *, void
51             *))(Point_cmpDistanceTarget, &target));
52     printf("Distance ");
53     Point_afficher(&target);
54     printf(" : \n");
55     for(i = 0; i < 5; ++i) {
56         Point_afficher(points + i);
57         printf(" %g\n", Point_distance(points + i, &target));
58     }
59     exit(EXIT_SUCCESS);
}
```

```
1  /* +-----+ */
2  /* / FICHIER : point.h / */
3  /* +-----+ */
4  #ifndef DEF_HEADER_POINT
5  #define DEF_HEADER_POINT
6
7  typedef struct Point Point;
8  struct Point {
9      double x;
10     double y;
11 };
12
13 Point Point_creer(double x, double y);
14
15 void Point_afficher(const Point * p);
16
17 double Point_distance(const Point * first, const Point * second);
18
19 #endif
```

```
1  /* +-----+ */
2  /* / FICHIER : point.c / */
3  /* +-----+ */
4  #include "point.h"
5
6  #include <stdio.h>
7  #include <math.h>
```

```

8 Point Point_creer(double x, double y) {
9     Point p = {x, y};
10    return p;
11 }
12
13
14 void Point_afficher(const Point * p) {
15     printf("(%.g, %.g)", p->x, p->y);
16 }
17
18 double Point_distance(const Point * first, const Point * second) {
19     return sqrt(pow(second->x - first->x, 2) + pow(second->y -
20     ↪ first->y, 2));
}

```

Correction 120 (★★★ Benchmark de qsort).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define TAILLE_DEMO      10
6 #define TAILLE_PERF       10000
7 #define TAILLE_BENCHMARK 100000000
8
9 void afficherInt(const void * elem) {
10    printf("%d", *(int *)elem);
11 }
12
13 int copierInt(void * dest, const void * src) {
14    *(int *)dest = *(const int *)src;
15    return 1;
16 }
17
18 void randomInt(void * elem) {
19    *(int *)elem = rand() % 1000000000;
20 }
21
22 int cmpInt(const void * first, const void * second) {
23    return *(int *)first - *(int *)second;
}

```

```
24 }
25
26 /* afficher_tableau affiche la liste de ses éléments au format */
27 /* [e1, e2, ..., eN] */ */
28 void afficher_tableau(const void * values, size_t elem, int
29   → taille, void (*afficherElement)(const void *));
30 /* copie_tableau recopie les éléments de source dans */
31 /* destination */ */
32 void copier_tableau(void * destination, const void * source,
33   → size_t elem, int taille, int (*copierElement)(void *, const
34   → void *));
35 /* aleatoire remplit un tableau par des valeurs aléatoires */
36 void aleatoire(void * valeurs, size_t elem, int taille, void
37   → (*randomElement)(void *));
38 /* swap échange les valeurs référencées par deux adresses */
39 void swap(void * a, void * b, size_t elem);
40 /* trier_algo est une implémentation du tri par tas */
41 void trier_algo(void * valeurs, size_t elem, int taille, int
42   → (*cmpElements)(const void *, const void *));
43 /* trier_eleve est votre méthode pour trier un tableau */
44 void trier_eleve(void * valeurs, size_t elem, int taille, int
45   → (*cmpElements)(const void *, const void *));
46 /* verif_trier renvoie 1 si le tableau est trié et 0 sinon */
47 int verif_trier(const void * valeurs, size_t elem, int taille, int
48   → (*cmpElements)(const void *, const void *));
49
50 int main() {
51     srand(time(NULL));
52     int * valeurs = (int *)malloc(TAILLE_BENCHMARK * sizeof(int));
53     int * methode_algo = (int *)malloc(TAILLE_BENCHMARK *
54       → sizeof(int));
55     int * methode_qsort = (int *)malloc(TAILLE_BENCHMARK *
56       → sizeof(int));
57     clock_t start, stop;
58
59     aleatoire(valeurs, sizeof(int), TAILLE_DEMO, randomInt);
60     copier_tableau(methode_algo, valeurs, sizeof(int), TAILLE_DEMO,
61       → copierInt);
62     copier_tableau(methode_qsort, valeurs, sizeof(int), TAILLE_DEMO,
63       → copierInt);
64     printf("non trié : ");
```

```

54 afficher_tableau(valeurs, sizeof(int), TAILLE_DEMO,
55     ↳ afficherInt);
56 trier_algo(methode_algo, sizeof(int), TAILLE_DEMO, cmpInt);
57 printf("methode_algo : ");
58 afficher_tableau(methode_algo, sizeof(int), TAILLE_DEMO,
59     ↳ afficherInt);
60 qsort(methode_qsort, TAILLE_DEMO, sizeof(int), cmpInt);
61 printf("methode_qsort : ");
62 afficher_tableau(methode_qsort, sizeof(int), TAILLE_DEMO,
63     ↳ afficherInt);

64 aleatoire(valeurs, sizeof(int), TAILLE_PERF, randomInt);
65 copier_tableau(methode_algo, valeurs, sizeof(int), TAILLE_PERF,
66     ↳ copierInt);
67 copier_tableau(methode_qsort, valeurs, sizeof(int), TAILLE_PERF,
68     ↳ copierInt);
69 printf("tableau de taille %d :\n", TAILLE_PERF);
70 start = clock();
71 trier_algo(methode_algo, sizeof(int), TAILLE_PERF, cmpInt);
72 stop = clock();
73 double temps_algo = (stop - start);
74 printf("methode_algo %s\n", verif_trier(methode_algo,
75     ↳ sizeof(int), TAILLE_PERF, cmpInt) ? "est trié" : "n'est pas
76     ↳ trié");
77 printf("Temps écoulé : %g s\n", temps_algo / CLOCKS_PER_SEC);
78 start = clock();
79 qsort(methode_qsort, TAILLE_PERF, sizeof(int), cmpInt);
80 stop = clock();
81 double temps_qsort = (stop - start);
82 printf("methode_qsort %s\n", verif_trier(methode_qsort,
83     ↳ sizeof(int), TAILLE_PERF, cmpInt) ? "est trié" : "n'est pas
84     ↳ trié");
85 printf("Temps écoulé : %g s\n", temps_qsort / CLOCKS_PER_SEC);
86 printf("Soit une différence qsort / algo de l'ordre d'un facteur
87     ↳ %g\n", temps_qsort / temps_algo);

88 aleatoire(valeurs, sizeof(int), TAILLE_BENCHMARK, randomInt);
89 copier_tableau(methode_algo, valeurs, sizeof(int),
90     ↳ TAILLE_BENCHMARK, copierInt);
91 copier_tableau(methode_qsort, valeurs, sizeof(int),
92     ↳ TAILLE_BENCHMARK, copierInt);

```

```
83 printf("tableau de taille %d :\n", TAILLE_BENCHMARK);
84 start = clock();
85 trier_algo(methode_algo, sizeof(int), TAILLE_BENCHMARK, cmpInt);
86 stop = clock();
87 temps_algo = (stop - start);
88 printf("methode_algo %s\n", verif_trier(methode_algo,
89     ↪ sizeof(int), TAILLE_BENCHMARK, cmpInt) ? "est trié" : "n'est
90     ↪ pas trié");
91 printf("Temps écoulé : %g s\n", temps_algo / CLOCKS_PER_SEC);
92 start = clock();
93 qsort(methode_qsort, TAILLE_BENCHMARK, sizeof(int), cmpInt);
94 stop = clock();
95 temps_qsort = (stop - start);
96 printf("methode_qsort %s\n", verif_trier(methode_qsort,
97     ↪ sizeof(int), TAILLE_BENCHMARK, cmpInt) ? "est trié" : "n'est
98     ↪ pas trié");
99 printf("Temps écoulé : %g s\n", temps_qsort / CLOCKS_PER_SEC);
100 printf("Soit une différence qsort / algo de l'ordre d'un facteur
101     ↪ %g\n", temps_qsort / temps_algo);
102 free(valeurs);
103 free(methode_algo);
104 free(methode_qsort);
105 exit(EXIT_SUCCESS);
106 }

107 void afficher_tableau(const void * values, size_t elem, int
108     ↪ taille, void (*afficherElement)(const void *)) {
109     int i;
110     printf("[");
111     for(i = 0; i < taille; ++i) {
112         if(i) printf(", ");
113         afficherElement((unsigned char *)values + elem * i);
114     }
115     printf("]\n");
116 }

117 void copier_tableau(void * destination, const void * source,
118     ↪ size_t elem, int taille, int (*copierElement)(void *, const
119     ↪ void *)) {
120     int i;
121     for(i = 0; i < taille; ++i) {
```

```

116     copierElement((unsigned char *)destination + elem * i,
117     ↪   (unsigned char *)source + elem * i);
118 }
119 }
120
121 void aleatoire(void * valeurs, size_t elem, int taille, void
122     ↪   (*randomElement)(void *)) {
123     int i;
124     for(i = 0; i < taille; ++i) {
125         randomElement((unsigned char *)valeurs + elem * i);
126     }
127 }
128
129 void swap(void * a, void * b, size_t elem) {
130     int i;
131     for(i = 0; i < elem; ++i) {
132         *((unsigned char *)a + i) ^= *((unsigned char *)b + i);
133         *((unsigned char *)b + i) ^= *((unsigned char *)a + i);
134         *((unsigned char *)a + i) ^= *((unsigned char *)b + i);
135     }
136 }
137
138 void trier_algo(void * valeurs, size_t elem, int taille, int
139     ↪   (*cmpElements)(const void *, const void *)) {
140     int indice;
141     int current;
142     int parent;
143     for(indice = 1; indice < taille; ++indice) {
144         current = indice;
145         parent = (current - 1) / 2;
146         while((current != 0) && (cmpElements((unsigned char *)valeurs
147             ↪ + elem * current, (unsigned char *)valeurs + elem *
148             ↪ parent) > 0)) {
149             swap((unsigned char *)valeurs + elem * current, (unsigned
150                 ↪ char *)valeurs + elem * parent, elem);
151             current = parent;
152             parent = (current - 1) / 2;
153         }
154     }
155     for(indice = taille - 1; indice >= 1; --indice) {
156         swap((unsigned char *)valeurs + elem * indice, valeurs, elem);

```

```
151     parent = 0;
152     while(parent * 2 + 1 < indice) {
153         if((parent * 2 + 2 >= indice) || (cmpElements((unsigned char
154             *)valeurs + elem * (parent * 2 + 1), (unsigned char
155             *)valeurs + elem * (parent * 2 + 2)) > 0)) {
156             current = parent * 2 + 1;
157         } else {
158             current = parent * 2 + 2;
159         }
160         if(cmpElements((unsigned char *)valeurs + elem * current,
161             (*unsigned char *)valeurs + elem * parent) < 0) {
162             break;
163         }
164     }
165 }
166
167 int verif_trier(const void * valeurs, size_t elem, int taille, int
168     (*cmpElements)(const void *, const void *)) {
169     int i;
170     for(i = 1; i < taille; ++i) {
171         if(cmpElements((unsigned char *)valeurs + elem * (i - 1),
172             (*unsigned char *)valeurs + elem * i) > 0)
173             return 0;
174     }
175     return 1;
176 }
```

Correction 121 (★★★ Trier des pointeurs de fonctions).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define ADDF(f) {f, #f}
5
6 typedef struct FName FName;
7 struct FName {
```

```
8   float (*f)(float);
9   char * name;
10 }
11
12 float carre(float x) {
13     return x * x;
14 }
15
16 float cube(float x) {
17     return x * x * x;
18 }
19
20 float inverse(float x) {
21     return 1.f / x;
22 }
23
24 float oppose(float x) {
25     return -x;
26 }
27
28 int fcmp(FName * first, FName * second, float * target) {
29     float d = first->f(*target) - second->f(*target);
30     if(d < 0) return -1;
31     else if(d > 0) return 1;
32     return 0;
33 }
34
35 int main() {
36     FName fonctions[] = {
37         ADDF(carre),
38         ADDF(cube),
39         ADDF(inverse),
40         ADDF(oppose),
41         ADDF(NULL)
42     };
43     float targets[] = {
44         -2.f,
45         -0.5f,
46         0.5f,
47         2.f
48     };
}
```

```
49 int i, j;
50 for(i = 0; i < 4; ++i) {
51     qsort_r(fonctions, 4, sizeof(FName), fcmp, targets + i);
52     printf("for %g :\n", targets[i]);
53     for(j = 0; j < 4; ++j) {
54         printf(" - %s(%g) = %g\n", fonctions[j].name, targets[i],
55                fonctions[j].f(targets[i]));
56     }
57     exit(EXIT_SUCCESS);
58 }
```

Correction 122 (*** Changer de langue).

```
1 executable :
2     gcc -o executable main.c -ldl
3     gcc -c fr.c -fPIC
4     gcc -shared -o fr.so fr.o
5     rm fr.o
6     gcc -c en.c -fPIC
7     gcc -shared -o en.so en.o
8     rm en.o
9
10 clean :
11     rm executable
12     rm fr.so en.so
```

```
1 /* +-----+ */
2 /* / FICHIER : main.c / */
3 /* +-----+ */
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <dlfcn.h>
7
8 typedef struct Langage Langage;
9 struct Langage {
10     void (*saluer)();
11     void (*quit)();
12     void * dlptr;
```

```

13 };
14
15 Langage Langage_charger(const char * path) {
16     void * dlptra = NULL;
17     Langage lang;
18     if((dlptr = dlopen(path, RTLD_NOW)) == NULL) {
19         fprintf(stderr, "Erreur d'ouverture du plugin \"%s\"\n", path,
20             dlerror());
21         exit(EXIT_FAILURE);
22     }
23     if((lang.saluer = dlsym(dlptra, "saluer")) == NULL) {
24         dlclose(dlptra);
25         fprintf(stderr, "Erreur de lecture de \"saluer\" du plugin
26             \"%s\"\n", path, dlerror());
27         exit(EXIT_FAILURE);
28     }
29     if((lang.quit = dlsym(dlptra, "quit")) == NULL) {
30         dlclose(dlptra);
31         fprintf(stderr, "Erreur de lecture de \"quit\" du plugin
32             \"%s\"\n", path, dlerror());
33         exit(EXIT_FAILURE);
34     }
35     printf("Chargement de la langue depuis \"%s\"\n", path);
36     lang.dlptra = dlptra;
37     return lang;
38 }
39
40 void Langage_liberer(Langage * lang) {
41     if(lang->dlptr) {
42         dlclose(lang->dlptr);
43         lang->dlptr = NULL;
44     }
45 }
46
47 int main(int argc, char * argv[]) {
48     const char * lang_path = argc > 1 ? argv[1] : "fr.so";
49     Langage lang = Langage_charger(lang_path);
50     lang.saluer();
51     lang.quit();
52     Langage_liberer(&lang);
53     exit(EXIT_SUCCESS);
54 }
```

51

```
}
```

```
1  /* +-----+ */
2  /* / FICHIER : fr.c / */
3  /* +-----+ */
4  #include <stdio.h>
5
6  void saluer() {
7      printf("Bonjour !\n");
8  }
9
10 void quitter() {
11     printf("Au revoir.\n");
12 }
```

```
1  /* +-----+ */
2  /* / FICHIER : en.c / */
3  /* +-----+ */
4  #include <stdio.h>
5
6  void saluer() {
7      printf("Hello !\n");
8  }
9
10 void quitter() {
11     printf("Bye.\n");
12 }
```

Correction 123 (★★★★ Simulation de combattants génériques).

```
1 CC= gcc
2 CFLAGS= -O2 -Wall -Werror -ansi
3 CLIBS= -lm -ldl
4 EXE= executable
5 OBJ= obj/
6 SRC= src/
7 INCL= include/
8 PLUGC= plugins/src/
```

```

9 PLUGL= plugins/lib/
10 FILEC:= $(wildcard $(SRC)*.c)
11 FILEH:= $(wildcard $(INCL)*.h)
12 FILEO:= $(patsubst $(SRC)%.*.c,$(OBJ)%.*.o,$(FILEC))
13 FILEPC:= $(wildcard $(PLUGC)*.*.c)
14 FILEPL:= $(patsubst $(PLUGC)%.*.c,$(PLUGL)%.*.so,$(FILEPC))
15 DFLAGS= -Wl,--export-dynamic
16
17 $(EXE) : $(FILEO)
18     $(CC) $(CFLAGS) -o $@ $^ $(DFLAGS) $(CLIBS)
19
20 $(OBJ)main.o : $(SRC)main.c $(FILEH)
21     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
22     $(CC) $(CFLAGS) -o $@ -c $<
23
24 $(OBJ)%.*.o : $(SRC)%.*.c $(INCL)%.*.h
25     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
26     $(CC) $(CFLAGS) -o $@ -c $<
27
28 plugin :
29     @if [ ! -d "$(PLUGL)" ]; then mkdir $(PLUGL); fi;
30     echo "Generation of plugins"
31     for i in `ls $(PLUGC)*.*.c`; do $(CC) $(CFLAGS) -c $$i -fPIC ;
32     done
33     for i in `ls *.*.o`; do $(CC) $(CFLAGS) -shared -o
34     - $(PLUGL)$$i%.so $$i $(CLIBS); done
35     rm -rf *.o
36
37 clean :
38     rm -rf $(OBJ)*.*.o
39     rm -rf $(OBJ)
40     rm -rf $(EXE)
41
42 mrproper : clean
43     rm -rf $(PLUGO)*.*.o
44     rm -rf $(PLUGO)
45     rm -rf $(PLUGL)*.*.so

```

```

1 /* +-----+ */
2 /* / FICHIER : src/main.c / */

```

```
3  /* +-----+ */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <time.h>
7
8  #include "../include/Personnage.h"
9  #include "../include/Spell.h"
10 #include "../include/AI.h"
11
12 int main(int argc, char * argv[]) {
13
14     srand(time(NULL));
15
16     Personnage garde = Personnage_defensive_soldier();
17     Personnage_init(&garde);
18     printf("%s entre dans l'arène.\n", Personnage_nom(&garde));
19     Personnage_display(&garde);
20
21     int i;
22     int tour = 0;
23     for(i = 1; i < argc; ++i) {
24
25         Personnage adv = Personnage_charger(argv[i]);
26         Personnage_init(&adv);
27         printf("%s entre dans l'arène.\n", Personnage_nom(&adv));
28         Personnage_display(&adv);
29
30         for(; Personnage_est_vivant(&garde) &&
31             Personnage_est_vivant(&adv); ++tour) {
32             printf("">>> TOUR %d :\n", tour);
33             if(Personnage_est_plus_rapide(&garde, &adv)) {
34                 Personnage_action(&garde, &adv);
35                 Personnage_action(&adv, &garde);
36             } else {
37                 Personnage_action(&adv, &garde);
38                 Personnage_action(&garde, &adv);
39             }
40             Personnage_display(&garde);
41             Personnage_display(&adv);
42         }
43         if(! Personnage_est_vivant(&adv)) {
```

```

43     printf("%s est vaincu.\n", Personnage_nom(&adv));
44 }
45 Personnage_quit(&adv);
46 }
47 if(Personnage_est_vivant(&garde)) {
48     printf("%s a survécu !\n", Personnage_nom(&garde));
49 } else {
50     printf("%s est vaincu !\n", Personnage_nom(&garde));
51 }
52 Personnage_quit(&garde);
53 exit(EXIT_SUCCESS);
54 }
```

```

1 /* +-----+ */
2 /* / FICHIER : include/Personnage.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_PERSONNAGE
5 #define DEF_HEADER_PERSONNAGE
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <dlfcn.h>
10
11 typedef struct Stats Stats;
12 struct Stats {
13     int vie;
14     int atk;
15     int def;
16     int vit;
17 };
18
19 typedef struct Personnage Personnage;
20 struct Personnage {
21     char nom[21];
22     Stats start;
23     Stats current;
24     void (*coups[4])(Personnage *, Personnage *);
25     int (*choix)(const Personnage *, const Personnage *);
26     void * dlptra;
27 };
```

```
28 Personnage Personnage_charger(const char * path);
29
30 void Personnage_keep_in_bounds(Personnage * perso);
31
32 void Personnage_init(Personnage * perso);
33
34 void Personnage_quit(Personnage * perso);
35
36 void Personnage_display(const Personnage * perso);
37
38 int Personnage_est_vivant(const Personnage * perso);
39
40 const char * Personnage_nom(const Personnage * perso);
41
42 int Personnage_est_plus_rapide(const Personnage * self, const
43 → Personnage * other);
44
45 void Personnage_action(Personnage * self, Personnage * other);
46
47 Personnage Personnage_default_soldier();
48
49 Personnage Personnage_defensive_soldier();
50
51 #endif
```

```
1 /* +-----+ */
2 /* / FICHIER : src/Personnage.c / */
3 /* +-----+ */
4 #include "../include/Personnage.h"
5
6 #include "../include/Spell.h"
7 #include "../include/AI.h"
8
9 #include <string.h>
10
11 Personnage Personnage_charger(const char * path) {
12     void * dlptra = NULL;
13     Personnage (*instantiate)() = NULL;
14     if((dlptr = dlopen(path, RTLD_NOW)) == NULL) {
```

```

15     fprintf(stderr, "Erreur d'ouverture du plugin \"%s\"\n%s\n",
16             → path, dlerror());
17     exit(EXIT_FAILURE);
18 }
19 if((instantiate = (Personnage (*)())dlsym(dlptr,
20     → "Personnage_instantiate")) == NULL) {
21     dlclose(dlptr);
22     fprintf(stderr, "Erreur de lecture de
23         → \"Personnage_instantiate\" du plugin \"%s\"\n%s\n", path,
24         → dlerror());
25     exit(EXIT_FAILURE);
26 }
27 fprintf(stderr, "Lecture de \"Personnage_instantiate\" du plugin
28     → \"%s\"\n", path);
29 Personnage perso = instantiate();
30 fprintf(stderr, "Personnage instancié\n");
31 perso.dlptr = dlptr;
32 return perso;
33 }

34 void Personnage_keep_in_bounds(Personnage * perso) {
35     if(perso->current.vie < 0) {
36         perso->current.vie = 0;
37     }
38     if(perso->current.atk < 1) {
39         perso->current.atk = 1;
40     }
41     if(perso->current.def < 1) {
42         perso->current.def = 1;
43     }
44     if(perso->current.vit < 1) {
45         perso->current.vit = 1;
46     }
47     if(perso->current.vie > perso->start.vie) {
48         perso->current.vie = perso->start.vie;
49     }
50     if(perso->current.atk > 1000 * perso->start.atk) {
51         perso->current.atk = 1000 * perso->start.atk;
52     }
53     if(perso->current.def > 1000 * perso->start.def) {
54         perso->current.def = 1000 * perso->start.def;
55     }
56 }
```

```
51 }
52 if(perso->current.vit > 1000 * perso->start.vit) {
53     perso->current.vit = 1000 * perso->start.vit;
54 }
55 }

56 void Personnage_init(Personnage * perso) {
57     perso->current = perso->start;
58 }

59 void Personnage_quit(Personnage * perso) {
60     if(perso->dptr) {
61         dlclose(perso->dptr);
62         perso->dptr = NULL;
63     }
64 }

65 void Personnage_display(const Personnage * perso) {
66     printf("+-----+\n");
67     printf("| %20s |\n", perso->nom);
68     printf("| vie : %14d |\n", perso->current.vie);
69     printf("| atk : %14d |\n", perso->current.atk);
70     printf("| def : %14d |\n", perso->current.def);
71     printf("| vit : %14d |\n", perso->current.vit);
72     printf("+-----+\n");
73 }

74 int Personnage_est_vivant(const Personnage * perso) {
75     return perso->current.vie > 0;
76 }

77 const char * Personnage_nom(const Personnage * perso) {
78     return perso->nom;
79 }

80 int Personnage_est_plus_rapide(const Personnage * self, const
81     Personnage * other) {
82     return self->current.vit >= other->current.vit;
83 }

84 void Personnage_action(Personnage * self, Personnage * other) {
```

```
91     int choix = self->choix(self, other);
92     self->coups[choix](self, other);
93 }
94
95 Personnage Personnage_default_soldier() {
96     return (Personnage) {
97         .nom = "Soldat",
98         .start = (Stats) {
99             .vie = 100,
100            .atk = 50,
101            .def = 50,
102            .vit = 10
103        },
104        .coups = {
105            &Spell_coup_simple,
106            &Spell_se_soigner,
107            &Spell_boost_atk,
108            &Spell_boost_def
109        },
110        .choix = &AI_hit_prior,
111        .dlptr = NULL
112    };
113 }
114
115 Personnage Personnage_defensive_soldier() {
116     return (Personnage) {
117         .nom = "Garde",
118         .start = (Stats) {
119             .vie = 120,
120             .atk = 30,
121             .def = 70,
122             .vit = 10
123         },
124         .coups = {
125             &Spell_coup_simple,
126             &Spell_se_soigner,
127             &Spell_boost_atk,
128             &Spell_boost_def
129        },
130         .choix = &AI_buff_prior,
131         .dlptr = NULL
```

```
132     };  
133 }
```

```
1  /* +-----+ */  
2  /* / FICHIER : include/Spell.h / */  
3  /* +-----+ */  
4  #ifndef DEF_HEADER_SPELL  
5  #define DEF_HEADER_SPELL  
6  
7  #include "Personnage.h"  
8  
9  void Spell_coup_simple(Personnage * self, Personnage * other);  
10  
11 void Spell_se_soigner(Personnage * self, Personnage * other);  
12  
13 void Spell_boost_atk(Personnage * self, Personnage * other);  
14  
15 void Spell_boost_def(Personnage * self, Personnage * other);  
16  
17 #endif
```

```
1  /* +-----+ */  
2  /* / FICHIER : src/Spell.c / */  
3  /* +-----+ */  
4  #include "../include/Spell.h"  
5  
6  void Spell_coup_simple(Personnage * self, Personnage * other) {  
7      int deg = self->current.atk / other->current.def;  
8      if(deg <= 0) {  
9          deg = 1;  
10     }  
11     other->current.vie -= deg;  
12     printf("%s frappe %s et lui inflige %d dégats.\n", self->nom,  
13            ↳ other->nom, deg);  
14     Personnage_keep_in_bounds(other);  
15 }  
16  
17 void Spell_se_soigner(Personnage * self, Personnage * other) {  
18     self->current.vie *= 1.10;  
19     printf("%s se soigne.\n", self->nom);
```

```

19 Personnage_keep_in_bounds(self);
20 }
21
22 void Spell_boost_atk(Personnage * self, Personnage * other) {
23     self->current.atk *= 1.20;
24     printf("%s aiguise sa lame.\n", self->nom);
25     Personnage_keep_in_bounds(self);
26 }
27
28 void Spell_boost_def(Personnage * self, Personnage * other) {
29     self->current.def *= 1.30;
30     printf("%s se renforce.\n", self->nom);
31     Personnage_keep_in_bounds(self);
32 }
```

```

1 /* +-----+ */
2 /* | FICHIER : include/AI.h | */
3 /* +-----+ */
4 #ifndef DEF_HEADER_AI
5 #define DEF_HEADER_AI
6
7 #include "Personnage.h"
8
9 int AI_hit_prior(const Personnage * self, const Personnage *
10    ↵ other);
11
12 int AI_buff_prior(const Personnage * self, const Personnage *
13    ↵ other);
14
15 #endif
```

```

1 /* +-----+ */
2 /* | FICHIER : src/AI.c | */
3 /* +-----+ */
4 #include "../include/AI.h"
5
6 int AI_hit_prior(const Personnage * self, const Personnage *
7    ↵ other) {
8     if(other->current.vie < 0.20 * other->start.vie) {
9         return 0;
```

```
9     }
10    if(self->current.vie < 0.25 * self->start.vie) {
11      return 1;
12    }
13    return 0;
14  }

15
16  int AI_buff_prior(const Personnage * self, const Personnage *
17    ↪ other) {
18    if(rand() % 5 == 0) {
19      return rand() % 4;
20    }
21    if(self->current.vie < 0.25 * self->start.vie) {
22      return 1;
23    }
24    if(self->current.def < 5. * self->start.def) {
25      return 3;
26    }
27    if(self->current.atk < 5. * self->start.atk) {
28      return 2;
29    }
30    return 0;
31 }
```

```
1  /* +-----+ */
2  /* | FICHIER : plugins/src/Balourd.c | */
3  /* +-----+ */
4  #include "../include/Personnage.h"
5  #include "../include/Spell.h"
6  #include "../include/AI.h"
7
8  #include <math.h>
9
10 void Spell_coup_lourd(Personnage * self, Personnage * other) {
11   int deg = pow((float)(self->current.atk) / other->current.def,
12    ↪ 2.f);
13   if(deg <= 0) {
14     deg = 1;
15   }
16   other->current.vie -= deg;
```

```

16     printf("%s frappe %s et lui inflige %d dégats.\n", self->nom,
17            ↪ other->nom, deg);
18     Personnage_keep_in_bounds(other);
19 }
20
21 int AI_bouriner(const Personnage * self, const Personnage * other)
22 {
23     if(rand() % 3 == 0) {
24         return 2;
25     }
26     if(other->current.vie < 0.20 * other->start.vie) {
27         return 0;
28     }
29     if(self->current.vie < 0.50 * self->start.vie) {
30         return 1;
31     }
32     return 0;
33 }
34
35 Personnage Personnage_instantiate() {
36     return (Personnage) {
37         .nom = "Balourd",
38         .start = (Stats) {
39             .vie = 120,
40             .atk = 100,
41             .def = 150,
42             .vit = 6
43         },
44         .coups = {
45             &Spell_coup_lourd,
46             &Spell_se_soigner,
47             &Spell_boost_atk,
48             &Spell_boost_def
49         },
50         .choix = &AI_bouriner
51     };
52 }
```

Correction 124 (★★★★★ HashMap et ArrayList génériques).

```
1 CC= gcc
2 CFLAGS= -O2 -Wall -Werror -ansi
3 CLIBS= -lm
4 EXE= executable
5 OBJ= obj/
6 SRC= src/
7 INCL= include/
8 FILEC:= $(wildcard $(SRC)*.c)
9 FILEH:= $(wildcard $(INCL)*.h)
10 FILEO:= $(patsubst $(SRC)%..c,$(OBJ)%..o,$(FILEC))
11
12 $(EXE) : $(FILEO)
13     $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
14
15 $(OBJ)main.o : $(SRC)main.c $(FILEH)
16     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
17     $(CC) $(CFLAGS) -o $@ -c $<
18
19 $(OBJ)%..o : $(SRC)%..c $(INCL)%..h
20     @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
21     $(CC) $(CFLAGS) -o $@ -c $<
22
23 clean :
24     rm -rf $(OBJ)*.o
25     rm -rf $(OBJ)
26     rm -rf $(EXE)
```

```
1 /* +-----+ */
2 /* / FICHIER : src/main.c / */
3 /* +-----+ */
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 #include "../include/arraylist.h"
9 #include "../include/hashmap.h"
10 #include "../include/types.h"
11
12 #define TEST
13 #undef TEST
```

```

14
15 #ifdef TEST
16 #define CAPACITY 5
17 #define ELEMENTS 20
18 #endif
19
20 #define ADD_TO_HASH(tab, size, type, id) \
21     liste = ArrayList_creer(TypeData##type(), size); \
22     for(i = 0; i < size; ++i) ArrayList_ajouter(liste, tab + i); \
23     HashMap_ajouter(map, keys + id, &liste);
24
25 int main() {
26 #ifdef TEST
27     HashMap * map = HashMap_creer(TypeDataInt(), TypeDataInt(),
28         → CAPACITY);
29     int vals[5] = {1, 17, 3, 42, 5};
30     ArrayList * liste =
31         → ArrayList_creer_depuis_valeurs(TypeDataInt(), 5, vals + 0,
32         → vals + 1, vals + 2, vals + 3, vals + 4);
33     long seed = time(NULL);
34     int value;
35     int count;
36     int tmp;
37     long i;
38     srand(seed);
39     for(i = 0; i < ELEMENTS; ++i) {
40         value = rand() % ELEMENTS;
41         count = 0;
42         HashMap_rechercher(map, &value, &count);
43         ++count;
44         HashMap_ajouter(map, &value, &count);
45     }
46     HashMap_afficher(stdout, map);
47     printf("\n");
48     ArrayList_afficher(stdout, liste);
49     printf("\n");
50     HashMap_free(&map);
51     ArrayList_free(&liste);
52 #else
53     HashMap * map = HashMap_creer(TypeDataString(),
54         → TypedataArrayList(), 100);

```

```
51 float fvals[7] = {4.5, 13.37, 1.2, 8.1, 99.9, 4.2, 1.2};
52 char * svals[2] = {"Hello", "ESGI"};
53 int ival[5] = {1, 17, 3, 42, 5};
54 long lvals[3] = {50000000000, 30000000000, -42000000000};
55 char cvals[4] = {'E', 'S', 'G', 'I'};
56 char * keys[5] = {"a virgule", "texte", "entiers", "grands",
57   ↵ "caracteres"};
58 int i;
59 ArrayList * liste = NULL;
60 ADD_TO_HASH(fvals, 7, Float, 0);
61 ADD_TO_HASH(svals, 2, String, 1);
62 ADD_TO_HASH(ival, 5, Int, 2);
63 ADD_TO_HASH(lvals, 3, Long, 3);
64 ADD_TO_HASH(cvals, 4, Char, 4);
65 HashMap_afficher(stdout, map);
66 printf("\n");
67 #endif
68 exit(EXIT_SUCCESS);
}
```

```
1 /* +-----+ */
2 /* / FICHIER : include/type_data.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_TYPE_DATA
5 #define DEF_HEADER_TYPE_DATA
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 typedef struct TypeData TypeData;
11 struct TypeData {
12     size_t size;
13     int (*copier)(void * dest, const void * src);
14     void (*free)(void * elem);
15     void (*afficher)(FILE * flow, const void * elem);
16     int (*compare)(const void * first, const void * second);
17     unsigned long (*hash)(const void * elem);
18     char * (*getType)();
19 };
20
```

21 #endif

```

1  /* +-----+ */
2  /* | FICHIER : include/types.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_TYPES
5  #define DEF_HEADER_TYPES

6
7  #include "type_int.h"
8  #include "type_float.h"
9  #include "type_char.h"
10 #include "type_long.h"
11 #include "type_string.h"

12
13 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : include/arraylist.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_LISTE
5  #define DEF_HEADER_LISTE

6
7  #include <stdio.h>
8  #include "type_data.h"

9
10 typedef struct ArrayList ArrayList;
11 struct ArrayList;

12
13 ArrayList * ArrayList_creer(TypeData type, int capacite);

14
15 ArrayList * ArrayList_creer_depuis_valeurs(TypeData type, int
16   ↳ capacite, ...);

17 void ArrayList_free(ArrayList ** arraylist);

18
19 int ArrayList_ajouter(ArrayList * liste, const void * valeur);

20
21 void ArrayList_afficher(FILE * flow, const ArrayList * liste);

22
23 int ArrayList_compter(const ArrayList * liste, const void *
24   ↳ valeur);

```

```
24
25 TypeData TypeDataArrayList();
26
27 #endif
```

```
1  /* +-----+ */
2  /* / FICHIER : src/arraylist.c / */
3  /* +-----+ */
4  #include "../include/arraylist.h"
5
6  #include <stdlib.h>
7  #include <string.h>
8  #include <stdarg.h>
9
10 struct ArrayList {
11     void * data;
12     int taille;
13     int capacite;
14     TypeData type;
15 };
16
17 static void * ArrayList_at(const ArrayList * liste, int id) {
18     return (unsigned char *) (liste->data) + (id *
19         (liste->type.size));
20 }
21
22 ArrayList * ArrayList_creer(TypeData type, int capacite) {
23     ArrayList * res = NULL;
24     if((res = (ArrayList *)malloc(sizeof(ArrayList))) == NULL) {
25         return NULL;
26     }
27     if(capacite > 0) {
28         if((res->data = malloc(capacite * type.size)) == NULL) {
29             free(res);
30             return NULL;
31         }
32     }
33     res->capacite = capacite;
34     res->taille = 0;
35     res->type = type;
```

```

35     return res;
36 }
37
38 ArrayList * ArrayList_creer_depuis_valeurs(TypeData type, int
39 → capacite, ...) {
40     ArrayList * liste = NULL;
41     if((liste = ArrayList_creer(type, capacite)) == NULL)
42         return NULL;
43     int i;
44     va_list ap;
45     va_start(ap, capacite);
46     for(i = 0; i < capacite; ++i) {
47         if(! ArrayList_ajouter(liste, va_arg(ap, const void *))) {
48             ArrayList_free(&liste);
49             return NULL;
50         }
51     }
52     va_end(ap);
53     return liste;
54 }
55
56 void ArrayList_free(ArrayList ** arraylist) {
57     if(arraylist == NULL || *arraylist == NULL) {
58         return;
59     }
60     if((*arraylist)->data != NULL) {
61         int i;
62         for(i = 0; i < (*arraylist)->taille; ++i) {
63             (*arraylist)->type.free(ArrayList_at(*arraylist, i));
64         }
65         free((*arraylist)->data);
66     }
67     free(*arraylist);
68     *arraylist = NULL;
69 }
70
71 static int ArrayList_update_capacity(ArrayList * liste) {
72     if(liste->taille < liste->capacite) {
73         return 1;
74     }
75     void * tmp = NULL;

```

```
75     int newCap = liste->capacite * 2 + 10;
76     if((tmp = realloc(liste->data, newCap * liste->type.size)) ==
77         NULL) {
78         return 0;
79     }
80     liste->data = tmp;
81     liste->capacite = newCap;
82     return 1;
83 }
84
85 int ArrayList_ajouter(ArrayList * liste, const void * valeur) {
86     if(! ArrayList_update_capacity(liste)) {
87         return 0;
88     }
89     liste->type.copier(ArrayList_at(liste, liste->taille), valeur);
90     ++(liste->taille);
91     return 1;
92 }
93
94 void ArrayList_afficher(FILE * flow, const ArrayList * liste) {
95     int i;
96     fprintf(flow, "ArrayList<%s> : [", liste->type.getType());
97     for(i = 0; i < liste->taille; ++i) {
98         if(i) fprintf(flow, ", ");
99         liste->type.afficher(flow, ArrayList_at(liste, i));
100    }
101    fprintf(flow, "]");
102 }
103
104 int ArrayList_compter(const ArrayList * liste, const void *
105                         valeur) {
106     int count = 0;
107     int i;
108     for(i = 0; i < liste->taille; ++i) {
109         if(liste->type.compare(ArrayList_at(liste, i), valeur))
110             ++count;
111     }
112     return count;
113 }
114
115 static int TypeDataArrayList_copier(ArrayList ** dest, const
116                                     ArrayList ** src);
```

```

114 static void TypeDataArrayList_free(ArrayList ** elem);
115 static void TypeDataArrayList_afficher(FILE * flow, const
116     → ArrayList ** elem);
116 static int TypeDataArrayList_compare(const ArrayList ** first,
117     → const ArrayList ** second);
117 static unsigned long TypeDataArrayList_hash(const ArrayList **
118     → elem);
118 static char * TypeDataArrayList_getType();
119
120 TypeData TypeDataArrayList() {
121     TypeData data;
122     data.size = sizeof(ArrayList *);
123     data.copier = (int (*)(void *, const void
124         → *))TypeDataArrayList_copier;
124     data.free = (void (*)(void *))TypeDataArrayList_free;
125     data.afficher = (void (*)(FILE *, const void
126         → *))TypeDataArrayList_afficher;
126     data.compare = (int (*)(const void *, const void
127         → *))TypeDataArrayList_compare;
127     data.hash = (unsigned long (*)(const void
128         → *))TypeDataArrayList_hash;
128     data.getType = TypeDataArrayList_getType;
129     return data;
129 }
130
131
132 static int TypeDataArrayList_copier(ArrayList ** dest, const
133     → ArrayList ** src) {
133     if((*dest = ArrayList_creer((*src)->type, (*src)->capacite)) ==
134         → NULL)
134         return 0;
135     memcpy((*dest)->data, (*src)->data, (*src)->capacite *
136         → (*src)->type.size);
136     (*dest)->taille = (*src)->taille;
137     return 1;
138 }
139
140 static void TypeDataArrayList_free(ArrayList ** elem) {
141     ArrayList_free(elem);
142 }
143
144 static void TypeDataArrayList_afficher(FILE * flow, const
144     → ArrayList ** elem) {

```

```
145     ArrayList_afficher(flow, *elem);
146 }
147
148 static int TypeDataArrayList_compare(const ArrayList ** first,
149                                     const ArrayList ** second) {
150     int i;
151     for(i = 0; i < (*first)->taille && i < (*second)->taille; ++i) {
152         if((*first)->type.compare(ArrayList_at(*first, i),
153                                     ArrayList_at(*second, i)) < 0) {
154             return -1;
155         } else if((*first)->type.compare(ArrayList_at(*first, i),
156                                     ArrayList_at(*second, i)) > 0) {
157             return 1;
158         }
159     }
160     if((*first)->taille < (*second)->taille) {
161         return -1;
162     } else if((*first)->taille > (*second)->taille) {
163         return 1;
164     }
165     return 0;
166 }
167
168 static unsigned long TypeDataArrayList_hash(const ArrayList **
169                                             elem) {
170     unsigned long c = 0;
171     int i;
172     for(i = 0; i < (*elem)->taille; ++i) {
173         c = c * 31 + (*elem)->type.hash(ArrayList_at(*elem, i));
174     }
175     return c;
176 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/hashmap.h / */
3  /* +-----+ */
```

```

4 #ifndef DEF_HEADER_HASHMAP
5 #define DEF_HEADER_HASHMAP
6
7 #include <stdio.h>
8 #include "type_data.h"
9
10 typedef struct HashMap HashMap;
11 struct HashMap;
12
13 HashMap * HashMap_creer(TypeData keyType, TypeData valueType, int
14   ↪ capacite);
15
16 void HashMap_free(HashMap ** hashmap);
17
18 int HashMap_ajouter(HashMap * hashmap, const void * key, const
19   ↪ void * valeur);
20
21 void HashMap_afficher(FILE * flow, const HashMap * hashmap);
22
23 int HashMap_rechercher(const HashMap * hashmap, const void *
24   ↪ valeur, void * res);
25
26 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : src/hashmap.c | */
3 /* +-----+ */
4 #include "../include/hashmap.h"
5
6 #include <stdlib.h>
7
8 typedef struct HashMapElement HME;
9 struct HashMapElement {
10   void * key;
11   void * value;
12   HME * next;
13 };
14
15 struct HashMap {
16   HME ** data;

```

```
17     int capacite;
18     TypeData keyType;
19     TypeData valueType;
20 };
21
22 static HME * HME_alloc(const void * key, const void * value, HME *
23 →   next, const TypeData * keyType, const TypeData * valueType);
24
25 static HME * HME_rechercher(HME * root, const void * key, const
26 →   TypeData * keyType, const TypeData * valueType);
27
28 static void HME_free(HME ** node, const TypeData * keyType, const
29 →   TypeData * valueType);
30
31 static HME * HME_alloc(const void * key, const void * value, HME *
32 →   next, const TypeData * keyType, const TypeData * valueType) {
33     HME * res = NULL;
34     if((res = (HME *)malloc(sizeof(HME))) == NULL) {
35         return NULL;
36     }
37     res->key = malloc(keyType->size);
38     keyType->copier(res->key, key);
39     res->value = malloc(valueType->size);
40     valueType->copier(res->value, value);
41     res->next = next;
42     return res;
43 }
44
45 static HME * HME_rechercher(HME * root, const void * key, const
46 →   TypeData * keyType, const TypeData * valueType) {
47     for(; root != NULL; root = root->next) {
48         if(keyType->compare(root->key, key) == 0) {
49             return root;
50         }
51     }
52     return NULL;
53 }
54
55 static void HME_free(HME ** node, const TypeData * keyType, const
56 →   TypeData * valueType) {
57     if(node == NULL || *node == NULL) {
```

```

52     return;
53 }
54 HME_free(&((*node)->next), keyType, valueType);
55 keyType->free((*node)->key);
56 valueType->free((*node)->value);
57 free((*node)->key);
58 free((*node)->value);
59 free(*node);
60 *node = NULL;
61 }

62 /* HashMap section */

63
64
65 HashMap * HashMap_creer(TypeData keyType, TypeData valueType, int
66 → capacite) {
66   HashMap * res = NULL;
67   if((res = (HashMap *)malloc(sizeof(HashMap))) == NULL) {
68     return NULL;
69   }
70   if((res->data = (HME **)calloc(capacite, sizeof(HME *)))
71     == NULL) {
72     free(res);
73     return NULL;
74   }
75   res->capacite = capacite;
76   res->keyType = keyType;
77   res->valueType = valueType;
78   return res;
79 }

80
81 void HashMap_free(HashMap ** hashmap) {
82   if(hashmap == NULL || *hashmap == NULL) {
83     return;
84   }
85   int i;
86   for(i = 0; i < (*hashmap)->capacite; ++i) {
87     HME_free((*hashmap)->data + i, &((*hashmap)->keyType),
88     → &((*hashmap)->valueType));
89   }
90   free((*hashmap)->data);
91   free(*hashmap);

```

```
91     *hashmap = NULL;
92 }
93
94 int HashMap_ajouter(HashMap * hashmap, const void * key, const
95     void * valeur) {
96     HME * current = NULL;
97     unsigned long hid = hashmap->keyType.hash(key)
98         % hashmap->capacite;
99     if((current = HME_rechercher(*(hashmap->data + hid), key,
100        ↪ &(hashmap->keyType), &(hashmap->valueType))) == NULL) {
101         if((current = HME_alloc(key, valeur, *(hashmap->data + hid),
102            ↪ &(hashmap->keyType), &(hashmap->valueType))) == NULL) {
103             return 0;
104         }
105         *(hashmap->data + hid) = current;
106     } else {
107         hashmap->valueType.copier(current->value, valeur);
108     }
109     return 1;
110 }
111
112 void HashMap_afficher(FILE * flow, const HashMap * hashmap) {
113     int i;
114     HME * current = NULL;
115     int count = 0;
116     fprintf(flow, "HashMap<%s, %s> : {", hashmap->keyType.getType(),
117         ↪ hashmap->valueType.getType());
118     for(i = 0; i < hashmap->capacite; ++i) {
119         for(current = *(hashmap->data + i); current != NULL; current =
120             ↪ current->next) {
121             if(count++) fprintf(flow, ", ");
122             hashmap->keyType.afficher(flow, current->key);
123             fprintf(flow, " : ");
124             hashmap->valueType.afficher(flow, current->value);
125         }
126     }
127     fprintf(flow, "}");
128 }
129
130 int HashMap_rechercher(const HashMap * hashmap, const void *
131     ↪ valeur, void * res) {
```

```

126     unsigned long hid = hashmap->keyType.hash(valeur)
127     % hashmap->capacite;
128     HME * current = HME_rechercher(*(hashmap->data + hid), valeur,
129     ↳ &(hashmap->keyType), &(hashmap->valueType));
130     return (current) ? (hashmap->valueType.copier(res,
131     ↳ current->value)) : 0;
    }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/type_char.h | */
3  /* +-----+ */
4  #ifndef DEF_HEADER_TYPE_CHAR
5  #define DEF_HEADER_TYPE_CHAR
6
7  #include "type_data.h"
8
9  TypeData TypeDataChar();
10
11 #endif
```

```

1  /* +-----+ */
2  /* | FICHIER : src/type_char.c | */
3  /* +-----+ */
4  #include "../include/type_char.h"
5
6  static int TypeDataChar_copier(char * dest, const char * src);
7  static void TypeDataChar_free(char * elem);
8  static void TypeDataChar_afficher(FILE * flow, const char * elem);
9  static int TypeDataChar_compare(const char * first, const char *
10   ↳ second);
11  static unsigned long TypeDataChar_hash(const char * elem);
12  static char * TypeDataChar_getType();
13
14  TypeData TypeDataChar() {
15      TypeData data;
16      data.size = sizeof(char);
17      data.copier = (int (*)(void *, const void
18       ↳ *))TypeDataChar_copier;
19      data.free = (void (*)(void *))TypeDataChar_free;
20      data.afficher = (void (*)(FILE *, const void
21       ↳ *))TypeDataChar_afficher;
```

```
19     data.compare = (int (*)(const void *, const void
20             ↵ *))TypeDataChar_compare;
21     data.hash = (unsigned long (*)(const void *))TypeDataChar_hash;
22     data.getType = TypeDataChar_getType;
23     return data;
24 }
25
26 static int TypeDataChar_copier(char * dest, const char * src) {
27     *dest = *src;
28     return 1;
29 }
30
31 static void TypeDataChar_free(char * elem) {
32     /* do nothing */
33 }
34
35 static void TypeDataChar_afficher(FILE * flow,
36         const char * elem) {
37     fprintf(flow, "\'%c\'", *elem);
38 }
39
40 static int TypeDataChar_compare(const char * first,
41         const char * second) {
42     return *first - *second;
43 }
44
45 static unsigned long TypeDataChar_hash(const char * elem) {
46     return *elem;
47 }
48
49 static char * TypeDataChar_getType() {
50     return "char";
51 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/type_float.h / */
3  /* +-----+ */
4  #ifndef DEF_HEADER_TYPE_FLOAT
5  #define DEF_HEADER_TYPE_FLOAT
6
```

```

7 #include "type_data.h"
8
9 TypeData TypeDataFloat();
10
11 #endif

```

```

1 /* +-----+ */
2 /* / FICHIER : src/type_float.c / */
3 /* +-----+ */
4 #include "../include/type_float.h"

5
6 #include <math.h>

7
8 static int TypeDataFloat_copier(float * dest, const float * src);
9 static void TypeDataFloat_free(float * elem);
10 static void TypeDataFloat_afficher(FILE * flow, const float *
11     elem);
12 static int TypeDataFloat_compare(const float * first, const float
13     * second);
14 static unsigned long TypeDataFloat_hash(const float * elem);
15 static char * TypeDataFloat_getType();

16 TypeData TypeDataFloat() {
17     TypeData data;
18     data.size = sizeof(float);
19     data.copier = (int (*)(void *, const void
20         *))TypeDataFloat_copier;
21     data.free = (void (*)(void *))TypeDataFloat_free;
22     data.afficher = (void (*)(FILE *, const void
23         *))TypeDataFloat_afficher;
24     data.compare = (int (*)(const void *, const void
25         *))TypeDataFloat_compare;
26     data.hash = (unsigned long (*)(const void *))TypeDataFloat_hash;
27     data.getType = TypeDataFloat_getType;
28     return data;
29 }

```

```

27 static int TypeDataFloat_copier(float * dest, const float * src) {
28     *dest = *src;
29     return 1;

```

```
30 }
31
32 static void TypeDataFloat_free(float * elem) {
33     /* do nothing */
34 }
35
36 static void TypeDataFloat_afficher(FILE * flow, const float *
37     → elem) {
38     fprintf(flow, "%g", *elem);
39 }
40
41 static int TypeDataFloat_compare(const float * first, const float
42     → * second) {
43     if(abs(*first - *second) < 1e-9) {
44         return 0;
45     }
46     if(*first < *second) {
47         return -1;
48     } else if(*first > *second) {
49         return 1;
50     }
51     return 0;
52 }
53
54 static unsigned long TypeDataFloat_hash(const float * elem) {
55     return *((int*)elem);
56 }
57
58 static char * TypeDataFloat_getType() {
59     return "float";
60 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/type_int.h / */
3  /* +-----+ */
4  #ifndef DEF_HEADER_TYPE_INT
5  #define DEF_HEADER_TYPE_INT
6
7  #include "type_data.h"
8
```

```

9  TypeData TypeDataInt();
10
11 #endif
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

```

```

/* +-----+ */
/* / FICHIER : src/type_int.c / */
/* +-----+ */
#include "../include/type_int.h"

static int TypeDataInt_copier(int * dest, const int * src);
static void TypeDataInt_free(int * elem);
static void TypeDataInt_afficher(FILE * flow, const int * elem);
static int TypeDataInt_compare(const int * first, const int *
→ second);
static unsigned long TypeDataInt_hash(const int * elem);
static char * TypeDataInt_getType();

TypeData TypeDataInt() {
    TypeData data;
    data.size = sizeof(int);
    data.copier = (int (*)(void *, const void *))TypeDataInt_copier;
    data.free = (void (*)(void *))TypeDataInt_free;
    data.afficher = (void (*)(FILE *, const void
→ *))TypeDataInt_afficher;
    data.compare = (int (*)(const void *, const void
→ *))TypeDataInt_compare;
    data.hash = (unsigned long (*)(const void *))TypeDataInt_hash;
    data.getType = TypeDataInt_getType;
    return data;
}

static int TypeDataInt_copier(int * dest, const int * src) {
    *dest = *src;
    return 1;
}

static void TypeDataInt_free(int * elem) {
    /* do nothing */
}

```

```
34 static void TypeDataInt_afficher(FILE * flow, const int * elem) {
35     fprintf(flow, "%d", *elem);
36 }
37
38 static int TypeDataInt_compare(const int * first,
39     const int * second) {
40     return *first - *second;
41 }
42
43 static unsigned long TypeDataInt_hash(const int * elem) {
44     return *elem;
45 }
46
47 static char * TypeDataInt_getType() {
48     return "int";
49 }
```

```
1 /* +-----+ */
2 /* / FICHIER : include/type_long.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_TYPE_LONG
5 #define DEF_HEADER_TYPE_LONG
6
7 #include "type_data.h"
8
9 TypeData TypeDataLong();
10
11 #endif
```

```
1 /* +-----+ */
2 /* / FICHIER : src/type_long.c / */
3 /* +-----+ */
4 #include "../include/type_long.h"
5
6 static int TypeDataLong_copier(long * dest, const long * src);
7 static void TypeDataLong_free(long * elem);
8 static void TypeDataLong_afficher(FILE * flow, const long * elem);
9 static int TypeDataLong_compare(const long * first, const long *
10     second);
11 static unsigned long TypeDataLong_hash(const long * elem);
```

```

11 static char * TypeDataLong_getType();
12
13 TypeData TypeDataLong() {
14     TypeData data;
15     data.size = sizeof(long);
16     data.copier = (int (*)(void *, const void
17     *))(TypeDataLong_copier);
18     data.free = (void (*)(void *))TypeDataLong_free;
19     data.afficher = (void (*)(FILE *, const void
20     *))(TypeDataLong_afficher);
21     data.compare = (int (*)(const void *, const void
22     *))(TypeDataLong_compare);
23     data.hash = (unsigned long (*)(const void *))TypeDataLong_hash;
24     data.getType = TypeDataLong_getType;
25     return data;
26 }
27
28 static int TypeDataLong_copier(long * dest, const long * src) {
29     *dest = *src;
30     return 1;
31 }
32
33 static void TypeDataLong_free(long * elem) {
34     /* do nothing */
35 }
36
37 static void TypeDataLong_afficher(FILE * flow,
38     const long * elem) {
39     fprintf(flow, "%ld", *elem);
40 }
41
42 static int TypeDataLong_compare(const long * first, const long *
43     second) {
44     if(*first < *second) {
45         return -1;
46     } else if(*first > *second) {
47         return 1;
48     }
49     return 0;
50 }
```

```
48 static unsigned long TypeDataLong_hash(const long * elem) {
49     return *elem;
50 }
51
52 static char * TypeDataLong_getType() {
53     return "long";
54 }
```

```
1  /* +-----+ */
2  /* | FICHIER : include/type_string.h | */
3  /* +-----+ */
4 #ifndef DEF_HEADER_TYPE_STRING
5 #define DEF_HEADER_TYPE_STRING
6
7 #include "type_data.h"
8
9 TypeData TypeDataString();
10
11 #endif
```

```
1  /* +-----+ */
2  /* | FICHIER : src/type_string.c | */
3  /* +-----+ */
4 #include "../include/type_string.h"
5
6 #include <string.h>
7
8 static int TypeDataString_copier(char ** dest, const char ** src);
9 static void TypeDataString_free(char ** elem);
10 static void TypeDataString_afficher(FILE * flow, const char **
11     elem);
11 static int TypeDataString_compare(const char ** first, const char
12     ** second);
12 static unsigned long TypeDataString_hash(const char ** elem);
13 static char * TypeDataString_getType();
14
15 TypeData TypeDataString() {
16     TypeData data;
17     data.size = sizeof(char *);
18     data.copier = (int (*)(void *, const void
19         *))TypeDataString_copier;
```

```

19     data.free = (void (*)(void *))TypeDataString_free;
20     data.afficher = (void (*)(FILE *, const void
21         *))TypeDataString_afficher;
22     data.compare = (int (*)(const void *, const void
23         *))TypeDataString_compare;
24     data.hash = (unsigned long (*)(const void
25         *))TypeDataString_hash;
26     data.getType = TypeDataString_getType;
27     return data;
28 }
29
30 static int TypeDataString_copier(char ** dest, const char ** src)
31 {
32     if(*src == NULL) {
33         *dest = NULL;
34         return 1;
35     }
36     if((*dest = (char *)malloc((1 + strlen(*src)) * sizeof(char)))
37         == NULL) {
38         return 0;
39     }
40     strcpy(*dest, *src);
41     return 1;
42 }
43
44 static void TypeDataString_free(char ** elem) {
45     if(*elem != NULL) {
46         free(*elem);
47         *elem = NULL;
48     }
49 }
50
51 static void TypeDataString_afficher(FILE * flow, const char **
52     elem) {
53     if(*elem == NULL)
54         return;
55     fprintf(flow, "\\\"%s\\\"", *elem);
56 }
57
58 static int TypeDataString_compare(const char ** first, const char
59     ** second) {
60 }
```

```
53     if(*first == NULL) {
54         return (*second == NULL) ? 0 : -1;
55     } else if(*second == NULL) {
56         return 1;
57     }
58     return strcmp(*first, *second);
59 }
60
61 static unsigned long TypeDataString_hash(const char ** elem) {
62     if(*elem == NULL) {
63         return 0;
64     }
65     unsigned long h = 0;
66     const char * current = NULL;
67     for(current = *elem; *current != '\0'; ++current) {
68         h = 31 * h + *current + 129;
69     }
70     return h;
71 }
72
73 static char * TypeDataString_getType() {
74     return "string";
75 }
```

Correction 125 (★★★★★ Tableaux dynamiques à n dimensions).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdarg.h>
4 #include <assert.h>
5
6 void * nmalloc(size_t atomic, int dimensions, ...) {
7     int dims[32];
8     int i;
9     va_list ap;
10    va_start(ap, dimensions);
11    /* Récupération des valeurs de chaque dimension : */
12    for(i = 0; i < dimensions; ++i) {
13        dims[i] = va_arg(ap, int);
14    }
```

```

15 va_end(ap);
16 size_t current = 1;
17 size_t len = 0;
18 /* Calcul de la taille à allouer en mémoire : */
19 /* - sauvegarde des valeurs */
20 /* - sauvegarde des pointeurs à chaque niveau */
21 for(i = 0; i < dimensions; ++i) {
22     current *= dims[i];
23     if(i == dimensions - 1) {
24         len += current * atomic;
25     } else {
26         len += current * sizeof(unsigned char *);
27     }
28 }
29 /* Allocation du bloc mémoire : */
30 unsigned char * data = NULL;
31 assert((data = (unsigned char *)malloc(len * sizeof(unsigned
32     → char)))) != NULL;
32 unsigned char * block = data;
33 unsigned char * map_block;
34 unsigned char * next_block = NULL;
35 len = 1;
36 /* Mise en relation de chaque pointeur du niveau supérieur avec
37     → les valeurs / pointeurs du niveau inférieur. */
38 for(i = 0; i < dimensions - 1; ++i) {
39     len *= dims[i];
40     if(i == dimensions - 2) {
41         current = atomic;
42     } else {
43         current = sizeof(unsigned char *);
44     }
45     next_block = block + len * sizeof(unsigned char *);
46     map_block = next_block;
47     for( ; block != next_block; block += sizeof(unsigned char *),
48         → map_block += dims[i + 1] * current) {
49         *((unsigned char **)block) = map_block;
50     }
51 }
52 return (void *)data;
}

```

```
53
54 void test_dim4() {
55     int nx = 4, ny = 5, nz = 6, nw = 7;
56     int **** voxel = nmalloc(sizeof(int), 4, nx, ny, nz, nw);
57     int i, j, k, l;
58     for(i = 0; i < nx; ++i) {
59         for(j = 0; j < ny; ++j) {
60             for(k = 0; k < nz; ++k) {
61                 for(l = 0; l < nw; ++l) {
62                     voxel[i][j][k][l] = i * ny * nz * nw + j * nz * nw + k *
63                     ↵ nw + l;
64                 }
65             }
66         }
67     }
68     for(i = 0; i < nx; ++i) {
69         for(j = 0; j < ny; ++j) {
70             for(k = 0; k < nz; ++k) {
71                 for(l = 0; l < nw; ++l) {
72                     assert(voxel[i][j][k][l] == i * ny * nz * nw + j * nz *
73                     ↵ nw + k * nw + l);
74                 }
75             }
76         }
77     }
78     free(voxel);
79     voxel = NULL;
80 }
81
82 void test_voxel() {
83     int nx = 5, ny = 7, nz = 10;
84     int *** voxel = nmalloc(sizeof(int), 3, nx, ny, nz);
85     int i, j, k;
86     for(i = 0; i < nx; ++i) {
87         for(j = 0; j < ny; ++j) {
88             for(k = 0; k < nz; ++k) {
89                 voxel[i][j][k] = i * ny * nz + j * nz + k;
90             }
91         }
92     }
93 }
```

```

92     for(j = 0; j < ny; ++j) {
93         for(k = 0; k < nz; ++k) {
94             assert(voxel[i][j][k] == i * ny * nz + j * nz + k);
95         }
96     }
97 }
98 free(voxel);
99 voxel = NULL;
100 }

101
102 void test_pixel() {
103     int nx = 10, ny = 20;
104     int ** pixel = nmalloc(sizeof(int), 2, nx, ny);
105     int i, j;
106     for(i = 0; i < nx; ++i) {
107         for(j = 0; j < ny; ++j) {
108             pixel[i][j] = i * ny + j;
109         }
110     }
111     for(i = 0; i < nx; ++i) {
112         for(j = 0; j < ny; ++j) {
113             assert(pixel[i][j] == i * ny + j);
114         }
115     }
116     free(pixel);
117     pixel = NULL;
118 }

119
120 void test_list() {
121     int nx = 10;
122     int * list = nmalloc(sizeof(int), 1, nx);
123     int i;
124     for(i = 0; i < nx; ++i) {
125         list[i] = i;
126     }
127     for(i = 0; i < nx; ++i) {
128         assert(list[i] == i);
129     }
130     free(list);
131     list = NULL;
132 }
```

```
133
134 int main() {
135     test_list();
136     test_pixel();
137     test_voxel();
138     test_dim4();
139     printf("Pas d'arrêt, vérifier avec valgrind.\n");
140     exit(EXIT_SUCCESS);
141 }
```

Correction 126 (∞ Algorithme d'exponentiation générique).

```
1 CC= gcc
2 CFLAGS= -O2 -g -Wno-unused-result -Wall -ansi -I./include
3 CLIBS= -lm -lgmp
4 EXE= executable
5 OBJ= obj/
6 SRC= src/
7 INCL= include/
8 FILEC:= $(wildcard $(SRC)*.c)
9 FILEH:= $(wildcard $(INCL)*.h)
10 FILEO:= $(patsubst $(SRC)%.c,$(OBJ)%.o,$(FILEC))

11 $(EXE) : $(FILEO)
12         $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)

14 $(OBJ)main.o : $(SRC)main.c $(FILEH)
15         @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
16         $(CC) $(CFLAGS) -o $@ -c $<

18 $(OBJ)%.o : $(SRC)%.c $(INCL)%.h
19         @if [ ! -d "$(OBJ)" ]; then mkdir $(OBJ); fi;
20         $(CC) $(CFLAGS) -o $@ -c $<

22 run : $(EXE)
23         ./$(EXE)

25 debug : $(EXE)
26         gdb ./$(EXE)
```

```

29 memory : $(EXE)
30     valgrind ./$(EXE)
31
32 clean :
33     rm -rf $(OBJ)*.o
34     rm -rf $(OBJ)
35     rm -rf $(EXE)

```

```

1  /* +-----+ */
2  /* | FICHIER : src/main.c | */
3  /* +-----+ */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <ReelField.h>
8 #include <ModularField.h>
9 #include <IntField.h>
10 #include <Algorithm.h>
11 #include <MatrixField.h>
12
13 #include <gmp.h>
14 #include <time.h>
15
16 int main() {
17     clock_t start;
18     clock_t end;
19
20     Field field = ReelField();
21
22     double v = 3.;
23     start = clock();
24     Algorithm_fast_power(&field, &v, &v, 10);
25     end = clock();
26     field.display(&field, stdout, &v);
27     printf("\n - (in %g s)\n", (double)(end - start) /
28           CLOCKS_PER_SEC);
29
30     field = ModularField(127);
31

```

```
32     unsigned int v = 42;
33     start = clock();
34     Algorithm_fast_power(&field, &v, &v, 125);
35     end = clock();
36     field.display(&field, stdout, &v);
37     printf("\n - (in %g s)\n", (double)(end - start) /
38           → CLOCKS_PER_SEC);
39 }
40
41 field = IntField();
42 {
43     mpz_t v;
44     mpz_init_set_ui(v, 2);
45     start = clock();
46     Algorithm_fast_power(&field, &v, &v, 10000);
47     end = clock();
48     field.display(&field, stdout, &v);
49     printf("\n - (in %g s)\n", (double)(end - start) /
50           → CLOCKS_PER_SEC);
51 }
52
53 field = Matrix2x2Field(ReelField());
54 {
55     double v[2][2] = {
56         {1., 1.},
57         {1., 0.}
58     };
59     start = clock();
60     Algorithm_fast_power(&field, v, v, 1000);
61     end = clock();
62     field.display(&field, stdout, v);
63     printf("\n - (in %g s)\n", (double)(end - start) /
64           → CLOCKS_PER_SEC);
65 }
66
67 field = Matrix2x2Field(IntField());
68 {
69     mpz_t v[2][2];
70     mpz_init_set_ui(v[0][0], 1); mpz_init_set_ui(v[0][1], 1);
71     mpz_init_set_ui(v[1][0], 1); mpz_init_set_ui(v[1][1], 0);
72     start = clock();
```

```

70     Algorithm_fast_power(&field, v, v, 20000 - 1);
71     end = clock();
72     field.display(&field, stdout, v);
73     printf("\n - (in %g s)\n", (double)(end - start) /
74         → CLOCKS_PER_SEC);
75
76     field = Matrix2x2Field(ModularField(127));
77 {
78     unsigned int v[2][2] = {
79         {1, 2},
80         {3, 4}
81     };
82     unsigned int a[2][2];
83     start = clock();
84     Algorithm_fast_power(&field, a, v, 42);
85     Algorithm_fast_power(&field, v, v, -42);
86     end = clock();
87     field.display(&field, stdout, v);
88     printf("\n");
89     field.display(&field, stdout, a);
90     printf("\n - (in %g s)\n", (double)(end - start) /
91         → CLOCKS_PER_SEC);
92     start = clock();
93     field.fill_multiplication(&field, v, v, a);
94     end = clock();
95     field.display(&field, stdout, v);
96     printf("\n - (in %g s)\n", (double)(end - start) /
97         → CLOCKS_PER_SEC);
98
99     field = Matrix2x2Field(Matrix2x2Field(ModularField(127)));
100 {
101     unsigned int v[2][2][2] = {
102         {
103             {{1, 2},
104             {5, 6}},
105             {{3, 4},
106             {7, 8}}
107         },

```

```
108     {
109         {{9, 10},
110          {13, 14}},
111
112         {{11, 12},
113          {15, 16}}
114     }
115 };
116 start = clock();
117 Algorithm_fast_power(&field, v, v, 42);
118 end = clock();
119 field.display(&field, stdout, v);
120 printf("\n - (in %g s)\n", (double)(end - start) /
121       → CLOCKS_PER_SEC);
122 }
123 exit(EXIT_SUCCESS);
```

```
1 /* +-----+ */
2 /* / FICHIER : include/Algorithm.h / */
3 /* +-----+ */
4
5 #ifndef DEF_HEADER_GENERIC_ALGORITHM
6 #define DEF_HEADER_GENERIC_ALGORITHM
7
8 #include <GenericField.h>
9
10 void Algorithm_naive_power(Field * field, void * result, void *
11   → value, long exponent);
12
13 void Algorithm_fast_power(Field * field, void * result, void *
14   → value, long exponent);
15
16 #endif
```

```
1 /* +-----+ */
2 /* / FICHIER : src/Algorithm.c / */
3 /* +-----+ */
4
5 #include <Algorithm.h>
```

```

6
7 void Algorithm_naive_power(Field * field, void * result, void *
→ value, long exponent) {
8     void * used = malloc(field->size);
9     if(exponent >= 0) {
10         field->fill_copy(field, used, value);
11     } else {
12         field->fill_multiplication_inverse(field, used, value);
13         exponent *= -1;
14     }
15     field->fill_multiplication_neutral(field, result);
16     for( ; exponent > 0; --exponent) {
17         field->fill_multiplication(field, result, result, used);
18     }
19     free(used);
20 }
21
22 void Algorithm_fast_power(Field * field, void * result, void *
→ value, long exponent) {
23     void * used = malloc(field->size);
24     field->fill_multiplication_neutral(field, used);
25     if(exponent >= 0) {
26         field->fill_copy(field, used, value);
27     } else {
28         field->fill_multiplication_inverse(field, used, value);
29         exponent *= -1;
30     }
31     field->fill_multiplication_neutral(field, result);
32     for( ; exponent > 0; exponent >>= 1) {
33         if(exponent & 1) {
34             field->fill_multiplication(field, result, result, used);
35         }
36         field->fill_multiplication(field, used, used, used);
37     }
38     free(used);
39 }
```

```

1 /* +-----+ */
2 /* / FICHIER : include/GenericField.h / */
3 /* +-----+ */
```

```
4  
5 #ifndef DEF_HEADER_GENERIC_FIELD  
6 #define DEF_HEADER_GENERIC_FIELD  
7  
8 #include <stdio.h>  
9 #include <stdlib.h>  
10  
11 typedef struct Field Field;  
12 struct Field {  
13     size_t size;  
14     void (*fill_addition_neutral)(Field *, void *);  
15     void (*fill_multiplication_neutral)(Field *, void *);  
16     void (*fill_addition)(Field *, void *, void *, void *);  
17     void (*fill_multiplication)(Field *, void *, void *, void  
18         *);  
19     void (*fill_copy)(Field *, void *, void *);  
20     void (*fill_addition_inverse)(Field *, void *, void *);  
21     void (*fill_multiplication_inverse)(Field *, void *, void  
22         *);  
23     size_t (*display)(Field *, FILE *, void *);  
24     void * userData;  
25 };  
#endif
```

```
1 /* +-----+ */  
2 /* / FICHIER : include/IntField.h / */  
3 /* +-----+ */  
4  
5 #ifndef DEF_HEADER_INT_FIELD  
6 #define DEF_HEADER_INT_FIELD  
7  
8 #include <GenericField.h>  
9  
10 Field IntField();  
11  
12 #endif
```

```

1  /* +-----+ */
2  /* / FICHIER : src/IntField.c / */
3  /* +-----+ */
4
5 #include <IntField.h>
6
7 #include <stdio.h>
8 #include <gmp.h>
9
10 void IntField_fill_addition_neutral(Field * field, mpz_t * value)
11   {
12     mpz_init_set_ui(*value, 0);
13   }
14
15 void IntField_fill_multiplication_neutral(Field * field, mpz_t *
16   value) {
17   mpz_init_set_ui(*value, 1);
18 }
19
20 void IntField_fill_addition(Field * field, mpz_t * result, mpz_t *
21   first, mpz_t * second) {
22   mpz_add(*result, *first, *second);
23 }
24
25 void IntField_fill_multiplication(Field * field, mpz_t * result,
26   mpz_t * first, mpz_t * second) {
27   mpz_mul(*result, *first, *second);
28 }
29
30 void IntField_fill_copy(Field * field, mpz_t * result, mpz_t *
31   value) {
32   mpz_set(*result, *value);
33 }
34
35 void IntField_fill_addition_inverse(Field * field, mpz_t * result,
36   mpz_t * value) {
37   mpz_mul_si(*result, *value, -1);
38 }
39
40 void IntField_fill_multiplication_inverse(Field * field, mpz_t *
41   result, mpz_t * value) {
42 }
```

```
35     if(mpz_cmpabs_ui(*value, 1)) {
36         mpz_init_set_ui(*result, 1);
37     } else {
38         mpz_init_set_ui(*result, 0);
39     }
40 }
41
42 size_t IntField_display(Field * field, FILE * ostream, mpz_t *
43 ← value) {
44     return mpz_out_str(ostream, 10, *value);
45 }
46
47 Field IntField() {
48     Field field;
49     field.size = sizeof(mpz_t);
50     field.fill_addition_neutral = (void (*)(Field *, void
51 ← *))IntField_fill_addition_neutral;
52     field.fill_multiplication_neutral = (void (*)(Field *, void
53 ← *))IntField_fill_multiplication_neutral;
54     field.fill_addition = (void (*)(Field *, void *, void *, void
55 ← *))IntField_fill_addition;
56     field.fill_multiplication = (void (*)(Field *, void *, void *,
57 ← void *))IntField_fill_multiplication;
58     field.fill_copy = (void (*)(Field *, void *, void
59 ← *))IntField_fill_copy;
60     field.fill_addition_inverse = (void (*)(Field *, void *, void
61 ← *))IntField_fill_addition_inverse;
62     field.fill_multiplication_inverse = (void (*)(Field *, void *,
63 ← void *))IntField_fill_multiplication_inverse;
64     field.display = (size_t (*)(Field *, FILE *, void
65 ← *))IntField_display;
66     field.userData = NULL;
67     return field;
68 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/MatrixField.h / */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_MATRIX_FIELD
```

```

6 #define DEF_HEADER_MATRIX_FIELD
7
8 #include <GenericField.h>
9
10 Field Matrix2x2Field(Field field);
11
12 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : src/MatrixField.c | */
3 /* +-----+ */
4
5 #include <MatrixField.h>
6
7 #include <stdio.h>
8
9 void Matrix2x2Field_fill_addition_neutral(Field * field, void *
→ value) {
10     Field * subfield = ((Field *) (field->userData));
11     unsigned char * cvalue = (unsigned char *) value;
12     int i;
13     for(i = 0; i < 4; ++i) {
14         subfield->fill_addition_neutral(subfield, (cvalue + i *
→ subfield->size));
15     }
16 }
17
18 void Matrix2x2Field_fill_multiplication_neutral(Field * field,
→ void * value) {
19     Field * subfield = ((Field *) (field->userData));
20     unsigned char * cvalue = (unsigned char *) value;
21     subfield->fill_multiplication_neutral(subfield, (cvalue + 0 *
→ subfield->size));
22     subfield->fill_addition_neutral(subfield, (cvalue + 1 *
→ subfield->size));
23     subfield->fill_addition_neutral(subfield, (cvalue + 2 *
→ subfield->size));
24     subfield->fill_multiplication_neutral(subfield, (cvalue + 3 *
→ subfield->size));
25 }

```

```
26
27 void Matrix2x2Field_fill_addition(Field * field, void * result,
28     → void * first, void * second) {
29     Field * subfield = ((Field *) (field->userData));
30     unsigned char * cresult = (unsigned char *) result;
31     unsigned char * cfirrst = (unsigned char *) first;
32     unsigned char * csecond = (unsigned char *) second;
33     int i;
34     for(i = 0; i < 4; ++i) {
35         subfield->fill_addition(subfield, (cresult + i *
36             → subfield->size), (cfirrst + i * subfield->size), (csecond +
37             → i * subfield->size));
38     }
39 }
40
41 void Matrix2x2Field_fill_multiplication(Field * field, void *
42     → result, void * first, void * second) {
43     Field * subfield = ((Field *) (field->userData));
44     unsigned char * cfirrst = (unsigned char *) first;
45     unsigned char * csecond = (unsigned char *) second;
46     int i, l, c;
47     void * tmp = malloc(subfield->size);
48     void * aux = malloc(field->size);
49     unsigned char * caux = (unsigned char *) aux;
50     subfield->fill_addition_neutral(subfield, tmp);
51     for(l = 0; l < 2; ++l) {
52         for(c = 0; c < 2; ++c) {
53             subfield->fill_addition_neutral(subfield, (caux + (l * 2 +
54                 → c) * subfield->size));
55             for(i = 0; i < 2; ++i) {
56                 subfield->fill_multiplication(subfield, tmp, (cfirrst + (l
57                     → * 2 + i) * subfield->size), (csecond + (i * 2 + c) *
58                     → subfield->size));
59                 subfield->fill_addition(subfield, (caux + (l * 2 + c) *
60                     → subfield->size), (caux + (l * 2 + c) *
61                     → subfield->size), tmp);
62             }
63         }
64     }
65     field->fill_copy(field, result, aux);
66     free(aux);
67 }
```

```

58     free(tmp);
59 }
60
61 void Matrix2x2Field_fill_copy(Field * field, void * result, void *
→   value) {
62     Field * subfield = ((Field *) (field->userData));
63     unsigned char * cresult = (unsigned char *) result;
64     unsigned char * cvalue = (unsigned char *) value;
65     int i;
66     for(i = 0; i < 4; ++i) {
67         subfield->fill_copy(subfield, (cresult + i * subfield->size),
→           (cvalue + i * subfield->size));
68     }
69 }
70
71 void Matrix2x2Field_fill_addition_inverse(Field * field, void *
→   result, void * value) {
72     Field * subfield = ((Field *) (field->userData));
73     unsigned char * cresult = (unsigned char *) result;
74     unsigned char * cvalue = (unsigned char *) value;
75     int i;
76     for(i = 0; i < 4; ++i) {
77         subfield->fill_addition_inverse(subfield, (cresult + i *
→           subfield->size), (cvalue + i * subfield->size));
78     }
79 }
80
81 void Matrix2x2Field_fill_multiplication_inverse(Field * field,
→   void * result, void * value) {
82     Field * subfield = ((Field *) (field->userData));
83     unsigned char * cresult = (unsigned char *) result;
84     unsigned char * cvalue = (unsigned char *) value;
85     void * determinant = malloc(subfield->size);
86     void * tmp = malloc(subfield->size);
87     subfield->fill_multiplication(subfield, determinant, (cvalue + 0
→       * subfield->size), (cvalue + 3 * subfield->size));
88     subfield->fill_multiplication(subfield, tmp, (cvalue + 1 *
→       subfield->size), (cvalue + 2 * subfield->size));
89     subfield->fill_addition_inverse(subfield, tmp, tmp);
90     subfield->fill_addition(subfield, tmp, determinant, tmp);
91     subfield->fill_multiplication_inverse(subfield, determinant,
→       tmp);

```

```
92     subfield->fill_copy(subfield, tmp, (cvalue + 3 *
93         ↵ subfield->size));
94     subfield->fill_copy(subfield, (cresult + 3 * subfield->size),
95         ↵ (cvalue + 0 * subfield->size));
96     subfield->fill_copy(subfield, (cresult + 0 * subfield->size),
97         ↵ tmp);
98     subfield->fill_addition_inverse(subfield, (cresult + 1 *
99         ↵ subfield->size), (cvalue + 1 * subfield->size));
100    subfield->fill_addition_inverse(subfield, (cresult + 2 *
101        ↵ subfield->size), (cvalue + 2 * subfield->size));
102    int i;
103    for(i = 0; i < 4; ++i) {
104        subfield->fill_multiplication(subfield, (cresult + i *
105            ↵ subfield->size), (cresult + i * subfield->size),
106            ↵ determinant);
107    }
108    free(determinant);
109    free(tmp);
110 }
111
112 size_t Matrix2x2Field_display(Field * field, FILE * ostream, void
113     ↵ * value) {
114     Field * subfield = ((Field *) (field->userData));
115     unsigned char * cvalue = (unsigned char *) value;
116     size_t count = 0;
117     count += fprintf(ostream, "{");
118     count += subfield->display(subfield, ostream, (cvalue + 0 *
119         ↵ subfield->size));
120     count += fprintf(ostream, ", ");
121     count += subfield->display(subfield, ostream, (cvalue + 1 *
122         ↵ subfield->size));
123     count += fprintf(ostream, ", ");
124     count += subfield->display(subfield, ostream, (cvalue + 2 *
125         ↵ subfield->size));
126     count += fprintf(ostream, ", ");
127     count += subfield->display(subfield, ostream, (cvalue + 3 *
128         ↵ subfield->size));
129     count += fprintf(ostream, "})");
130     return count;
131 }
```

```

121 Field Matrix2x2Field(Field subfield) {
122     Field field;
123     field.size = subfield.size * 4;
124     field.fill_addition_neutral =
125         → Matrix2x2Field_fill_addition_neutral;
126     field.fill_multiplication_neutral =
127         → Matrix2x2Field_fill_multiplication_neutral;
128     field.fill_addition = Matrix2x2Field_fill_addition;
129     field.fill_multiplication = Matrix2x2Field_fill_multiplication;
130     field.fill_copy = Matrix2x2Field_fill_copy;
131     field.fill_addition_inverse =
132         → Matrix2x2Field_fill_addition_inverse;
133     field.fill_multiplication_inverse =
134         → Matrix2x2Field_fill_multiplication_inverse;
135     field.display = Matrix2x2Field_display;
136     field.userData = malloc(sizeof(Field));
137     *((Field *) (field.userData)) = subfield;
138     return field;
139 }

```

```

1  /* +-----+ */
2  /* | FICHIER : include/ModularField.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_MODULAR_FIELD
6 #define DEF_HEADER_MODULAR_FIELD
7
8 #include <GenericField.h>
9
10 Field ModularField(unsigned int p);
11
12 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : src/ModularField.c | */
3  /* +-----+ */
4
5 #include <ModularField.h>
6
7 #include <stdio.h>

```

```
8 void ModularField_fill_addition_neutral(Field * field, unsigned
9   ↪ int * value) {
10    *value = 0;
11 }
12
13 void ModularField_fill_multiplication_neutral(Field * field,
14   ↪ unsigned int * value) {
15    *value = 1;
16 }
17
18 void ModularField_fill_addition(Field * field, unsigned int *
19   ↪ result, unsigned int * first, unsigned int * second) {
20    *result = (*first + *second) % *((unsigned int
21      ↪ *) (field->userData));
22 }
23
24 void ModularField_fill_multiplication(Field * field, unsigned int
25   ↪ * result, unsigned int * first, unsigned int * second) {
26    *result = (*first * *second) % *((unsigned int
27      ↪ *) (field->userData));
28 }
29
30 void ModularField_fill_copy(Field * field, unsigned int * result,
31   ↪ unsigned int * value) {
32    *result = *value;
33 }
34
35 void ModularField_fill_addition_inverse(Field * field, unsigned
36   ↪ int * result, unsigned int * value) {
37    *result = (-*value + *((unsigned int *) (field->userData))) %
38      ↪ *((unsigned int *) (field->userData));
39 }
40
41 void ModularField_fill_multiplication_inverse(Field * field,
42   ↪ unsigned int * result, unsigned int * value) {
43    long u0 = 1, v0 = 0;
44    long u1 = 0, v1 = 1;
45    long r0, r1;
46    long tmp;
47    r0 = *((unsigned int *) (field->userData));
```

```

39   r1 = *value;
40
41   long q;
42   while(r1 != 0) {
43     q = r0 / r1;
44     tmp = r1;
45     r1 = r0 - q * r1;
46     r0 = tmp;
47     tmp = u1;
48     u1 = u0 - q * u1;
49     u0 = tmp;
50     tmp = v1;
51     v1 = v0 - q * v1;
52     v0 = tmp;
53   }
54
55   *result = (unsigned int)(v0 + *((unsigned int
56   ↪   *)((field->userData))) % *((unsigned int
57   ↪   *)((field->userData)));
58 }
59
60 Field ModularField(unsigned int p) {
61   Field field;
62   field.size = sizeof(unsigned int);
63   field.fill_addition_neutral = (void (*)(Field *, void
64   ↪   *))ModularField_fill_addition_neutral;
65   field.fill_multiplication_neutral = (void (*)(Field *, void
66   ↪   *))ModularField_fill_multiplication_neutral;
67   field.fill_addition = (void (*)(Field *, void *, void *, void
68   ↪   *))ModularField_fill_addition;
69   field.fill_multiplication = (void (*)(Field *, void *, void *,
70   ↪   void *))ModularField_fill_multiplication;
71   field.fill_copy = (void (*)(Field *, void *, void
72   ↪   *))ModularField_fill_copy;
73   field.fill_addition_inverse = (void (*)(Field *, void *, void
74   ↪   *))ModularField_fill_addition_inverse;
75   field.fill_multiplication_inverse = (void (*)(Field *, void *,
76   ↪   void *))ModularField_fill_multiplication_inverse;

```

```
70     field.display = (size_t (*)(Field *, FILE *, void
    ↵   *))ModularField_display;
71     field.userData = malloc(sizeof(unsigned int));
72     *((unsigned int *) (field.userData)) = p;
73     return field;
74 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/ReelField.h / */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_REEL_FIELD
6 #define DEF_HEADER_REEL_FIELD
7
8 #include <GenericField.h>
9
10 Field ReelField();
11
12 #endif
```

```
1  /* +-----+ */
2  /* / FICHIER : src/ReelField.c / */
3  /* +-----+ */
4
5 #include <ReelField.h>
6
7 #include <stdio.h>
8
9 void ReelField_fill_addition(Field * field, double *
    ↵   value) {
10   *value = 0.;
11 }
12
13 void ReelField_fill_multiplication_neutral(Field * field, double *
    ↵   value) {
14   *value = 1.;
15 }
16
17 void ReelField_fill_addition(Field * field, double * result,
    ↵   double * first, double * second) {
```

```

18     *result = *first + *second;
19 }
20
21 void ReelField_fill_multiplication(Field * field, double * result,
22     → double * first, double * second) {
23     *result = *first * *second;
24 }
25
26 void ReelField_fill_copy(Field * field, double * result, double *
27     → value) {
28     *result = *value;
29 }
30
31 void ReelField_fill_addition_inverse(Field * field, double *
32     → result, double * value) {
33     *result = -*value;
34 }
35
36 void ReelField_fill_multiplication_inverse(Field * field, double *
37     → result, double * value) {
38     *result = 1. / *value;
39 }
40
41 Field ReelField() {
42     Field field;
43     field.size = sizeof(double);
44     field.fill_addition_neutral = (void (*)(Field *, void
45         → *))ReelField_fill_addition_neutral;
46     field.fill_multiplication_neutral = (void (*)(Field *, void
47         → *))ReelField_fill_multiplication_neutral;
48     field.fill_addition = (void (*)(Field *, void *, void *, void
49         → *))ReelField_fill_addition;
50     field.fill_multiplication = (void (*)(Field *, void *, void *,
51         → void *))ReelField_fill_multiplication;
52     field.fill_copy = (void (*)(Field *, void *, void
53         → *))ReelField_fill_copy;

```

```
49     field.fill_addition_inverse = (void (*)(Field *, void *, void
50         ↵ *))ReelField_fill_addition_inverse;
51     field.fill_multiplication_inverse = (void (*)(Field *, void *,
52         ↵ void *))ReelField_fill_multiplication_inverse;
53     field.display = (size_t (*)(Field *, FILE *, void
54         ↵ *))ReelField_display;
55     field.userData = NULL;
56     return field;
57 }
```

Correction 127 (∞ Space invanders modulaire).

```
1 CC= gcc
2 CFLAGS= -O2 -g -Wall -Werror -ansi -Wno-unused-result -I./include
3     ↵ -Wl,--export-dynamic
4 CLIBS= -lm -lSDL -lSDL_gfx -ldl
5 EXE= executable
6 OBJ= obj/
7 SRC= src/
8 INCL= include/
9 PLUG= plugins/
10 LIB= extensions/
11 FILEC:= $(wildcard $(SRC)*.c)
12 FILEH:= $(wildcard $(INCL)*.h)
13 FILEO:= $(patsubst $(SRC)%.c,$(OBJ)%.o,$(FILEC))
14 FILEP:= $(wildcard $(PLUG)*.c)
15 FILEL:= $(patsubst $(PLUG)%.c,$(LIB)%.so,$(FILEP))
16
17 $(EXE) : $(FILEO) $(FILEL)
18     $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
19
20 $(OBJ)main.o : $(SRC)main.c $(FILEH)
21     @if [ ! -d "$(OBJ)" ] ; then mkdir $(OBJ); fi;
22     $(CC) $(CFLAGS) -o $@ -c $<
23
24 $(OBJ)%.o : $(SRC)%.c $(INCL)%.h
25     @if [ ! -d "$(OBJ)" ] ; then mkdir $(OBJ); fi;
26     $(CC) $(CFLAGS) -o $@ -c $<
27
28 $(LIB)%.so : $(PLUG)%.c
```

```

28      @if [ ! -d "$(LIB)" ]; then mkdir $(LIB); fi;
29      $(CC) $(CFLAGS) -o tmp.o -c $< -fPIC
30      $(CC) -shared $(CFLAGS) -o $@ tmp.o $(CLIBS)
31      rm -rf tmp.o
32
33 run : $(EXE)
34     ./$(EXE)
35
36 debug : $(EXE)
37     gdb ./$(EXE)
38
39 memory : $(EXE)
40     valgrind ./$(EXE)
41
42 clean :
43     rm -rf $(OBJ)*.o
44     rm -rf $(OBJ)
45     rm -rf $(EXE)
46     rm -rf $(LIB)*.so
47     rm -rf $(LIB)

```

```

1  /* +-----+ */
2  /* / FICHIER : src/main.c / */
3  /* +-----+ */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <math.h>
8 #include <time.h>
9 #include <SDL/SDL.h>
10 #include <SDL/SDL_gfxPrimitives.h>
11 #include <Game.h>
12 #include <Component.h>
13
14 /* Paramètres de la fenêtre : */
15 const int largeur = 1200;
16 const int hauteur = 900;
17 const char * titre = "ESGI SpaceCraft";
18 Game game;
19

```

```
20 void affichage(SDL_Surface * ecran) {
21     unsigned long t = SDL_GetTicks();
22     int x, y, i;
23     /* Remplissage de l'écran par un gris foncé uniforme : */
24     SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 25, 25,
25             → 53));
25     for(i = 1; i < 100; ++i) {
26         x = 123 * i * i + 800 * sin((t + i * i * 100.) / 20000.);
27         y = 753 * i + t / 4. + 100 * cos((t + i * 100.) / 2000.);
28         x = ((x % largeur) + largeur) % largeur;
29         y = ((y % hauteur) + hauteur) % hauteur;
30         double coeff = 10. / (10. + i);
31         boxRGBA(ecran, x - 81 * coeff, y - 81 * coeff, x + 81 * coeff,
32             → y + 81 * coeff, 204, 204, 204, 3);
32         boxRGBA(ecran, x - 27 * coeff, y - 27 * coeff, x + 27 * coeff,
33             → y + 27 * coeff, 204, 204, 204, 9);
33         boxRGBA(ecran, x - 9 * coeff, y - 9 * coeff, x + 9 * coeff, y
34             → + 9 * coeff, 204, 204, 204, 27);
34         boxRGBA(ecran, x - 3 * coeff, y - 3 * coeff, x + 3 * coeff, y
35             → + 3 * coeff, 204, 204, 204, 81);
35         boxRGBA(ecran, x - 1, y - 1, x + 1, y + 1, 204, 204, 204,
36             → 255);
36     }
37 }
38
39 int main(int argc, char * argv[]) {
40     Ship * ship = Ship_load((argc > 1) ? argv[1] :
41         "extensions/Ship_basic.so", 1, Position_create(largeur / 2,
42             → 3 * hauteur / 4));
41     game = Game_create(largeur, hauteur, ship);
42     List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
43         "extensions/Ship_basic.so", 2, Position_create(largeur / 2,
44             → hauteur / 8)));
43     List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
44         "extensions/Ship_basic.so", 2, Position_create(largeur / 4,
45             → hauteur / 9)));
44     List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
45         "extensions/Ship_basic.so", 2, Position_create(3 * largeur /
46             → 4, hauteur / 9)));
45     List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
46         "extensions/Ship_tank.so", 2, Position_create(largeur / 4, 2
47             → * hauteur / 9)));
47 }
```

```

46 List_add(&(game.opponents), Ship_load((argc > 2) ? argv[2] :
47   ↵ "extensions/Ship_tank.so", 2, Position_create(3 * largeur /
48   ↵ 4, 2 * hauteur / 9)));
49 srand(time(NULL));
50 /* Création d'une fenêtre SDL : */
51 if(SDL_Init(SDL_INIT_EVERYTHING) != 0) {
52   fprintf(stderr, "Error in SDL_Init : %s\n", SDL_GetError());
53   exit(EXIT_FAILURE);
54 }
55 SDL_Surface * ecran = NULL;
56 if((ecran = SDL_SetVideoMode(largeur, hauteur, 32, SDL_HWSURFACE
57   ↵ | SDL_DOUBLEBUF)) == NULL) {
58   fprintf(stderr, "Error in SDL_SetVideoMode : %s\n",
59   ↵ SDL_GetError());
60   SDL_Quit();
61   exit(EXIT_FAILURE);
62 }
63 SDL_WM_SetCaption(titre, NULL);
64
65 int active = 1;
66 SDL_Event event;
67
68 while(active) {
69
70   affichage(ecran);
71   Game_draw(&game, ecran);
72   SDL_Flip(ecran);
73
74   while(SDL_PollEvent(&event)) {
75
76     switch(event.type) {
77       /* Utilisateur clique sur la croix de la fenêtre : */
78       case SDL_QUIT : {
79         active = 0;
80       } break;
81
82       /* Utilisateur enfonce une touche du clavier : */
83       case SDL_KEYDOWN : {
84         switch(event.key.keysym.sym) {
85           /* Touche Echap : */
86           case SDLK_ESCAPE : {

```

```
83         active = 0;
84     } break;

85
86     case SDLK_SPACE : {
87         game.player->control.ship_action = 1;
88     } break;

89
90     case SDLK_z :
91     case SDLK_UP : {
92         game.player->control.move_forward = 1;
93     } break;

94
95     case SDLK_s :
96     case SDLK_DOWN : {
97         game.player->control.move_backward = 1;
98     } break;

99
100    case SDLK_q :
101    case SDLK_LEFT : {
102        game.player->control.move_left = 1;
103    } break;

104
105    case SDLK_d :
106    case SDLK_RIGHT : {
107        game.player->control.move_right = 1;
108    } break;

109
110    default : break;
111 }
112 } break;

113
114 case SDL_KEYUP : {
115     switch(event.key.keysym.sym) {
116         case SDLK_SPACE : {
117             game.player->control.ship_action = 0;
118         } break;

119
120         case SDLK_z :
121         case SDLK_UP : {
122             game.player->control.move_forward = 0;
123         } break;
```

```

124
125     case SDLK_s :
126     case SDLK_DOWN : {
127         game.player->control.move_backward = 0;
128     } break;
129
130     case SDLK_q :
131     case SDLK_LEFT : {
132         game.player->control.move_left = 0;
133     } break;
134
135     case SDLK_d :
136     case SDLK_RIGHT : {
137         game.player->control.move_right = 0;
138     } break;
139
140     default : break;
141 }
142 } break;
143
144     default : break;
145 }
146 }
147 int delay = SDL_GetTicks() + 1000 / 60;
148 Game_play_step(&game);
149 delay -= SDL_GetTicks();
150 if(delay > 0) {
151     SDL_Delay(delay);
152 }
153 }
154
155     SDL_FreeSurface(ecran);
156     SDL_Quit();
157     exit(EXIT_SUCCESS);
158 }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/Component.h | */
3  /* +-----+ */
4
```

```
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_COMPONENT
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_COMPONENT
7
8 #include "Stats.h"
9 #include "Position.h"
10 #include "Projectile.h"
11 #include <SDL/SDL.h>
12
13 struct Ship;
14 struct Game;
15
16 typedef enum {
17     CT_NONE = 0,
18     CT_COCKPIT,
19     CT_ARMOR,
20     CT_WEAPON,
21     CT_PULSOR,
22     CT.Utility
23 } ComponentType;
24
25 typedef struct Component Component;
26 struct Component {
27     ComponentType type;
28     Stats stats;
29     int width;
30     int height;
31     int componentId;
32     Position position;
33     void (*action)(Component *, struct Ship *, struct Game *);
34     void (*draw)(Component *, SDL_Surface *, int, int, int, int);
35     void * userData;
36     void * dlptr;
37 };
38
39 Component * Component_alloc(ComponentType type, Stats stats,
40                             int width, int height, Position position);
41
42 void Component_free(Component * component);
43
44 Component * Component_load(const char * path, Position position);
45
```

46 #endif

```

1  /* +-----+ */
2  /* | FICHIER : src/Component.c | */
3  /* +-----+ */

4
5 #include "../include/Component.h"

6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <assert.h>
10 #include <dlfcn.h>

11
12 static int LastComponentId = 0;

13
14 Component * Component_alloc(ComponentType type, Stats stats,
15     int width, int height, Position position) {
16
17     Component * component = NULL;
18     assert(component = (Component *)malloc(sizeof(Component)));
19     component->type = type;
20     component->stats = stats;
21     component->width = width;
22     component->height = height;
23     component->componentId = ++LastComponentId;
24     component->position = position;
25     component->action = NULL;
26     component->draw = NULL;
27     component->userData = NULL;
28     component->dptr = NULL;
29     return component;
30 }
31
32 void Component_free(Component * component) {
33     if(component == NULL) {
34         return;
35     }
36     if(component->dptr) {
37         dlclose(component->dptr);
38     }

```

```
39     free(component->userData);
40     free(component);
41 }
42
43 Component * Component_load(const char * path, Position position) {
44     void * dlptra = NULL;
45     if((dlptr = dlopen(path, RTLD_NOW)) == NULL) {
46         fprintf(stderr, "Erreur d'ouverture du composant
47             ↪ \"%s\"\n%s\n", path, dlerror());
48         exit(EXIT_FAILURE);
49     }
50     Component * (*instantiate)(Position) = NULL;
51     if((instantiate = (Component * (*)(Position))dlsym(dlptra,
52             ↪ "Component_instantiate")) == NULL) {
53         fprintf(stderr, "Erreur de chargement du composant
54             ↪ \"%s\"\n%s\n", path, dlerror());
55         exit(EXIT_FAILURE);
56     }
57     Component * component = (*instantiate)(position);
58     component->dlptr = dlptra;
59     return component;
60 }
```

```
1  /* +-----+ */
2  /* / FICHIER : include/Game.h / */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_GAME
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_GAME
7
8 #include "Ship.h"
9 #include "Projectile.h"
10 #include <SDL/SDL.h>
11
12 typedef struct Game Game;
13 struct Game {
14     int width;
15     int height;
16     Ship * player;
17     List opponents;
```

```

18 };
19
20 Game Game_create(int width, int height, Ship * player);
21
22 void Game_free(Game * game);
23
24 void Game_draw(Game * game, SDL_Surface * ecran);
25
26 void Game_play_step(Game * game);
27
28 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : src/Game.c | */
3 /* +-----+ */
4
5 #include "../include/Game.h"
6
7 #include <stdlib.h>
8 #include <math.h>
9 #include <SDL/SDL_gfxPrimitives.h>
10
11 Game Game_create(int width, int height, Ship * player) {
12     Game game;
13     game.width = width;
14     game.height = height;
15     game.player = player;
16     game.opponents = List_empty();
17     return game;
18 }
19
20 void Game_free(Game * game) {
21     if(game == NULL) {
22         return;
23     }
24     Ship_free(game->player);
25     game->player = NULL;
26     List_free(&(game->opponents));
27 }
28

```

```
29 void Game_draw(Game * game, SDL_Surface * ecran) {
30     Ship_draw(game->player, ecran);
31
32     void process(Ship * ship) {
33         Ship_draw(ship, ecran);
34     };
35     List_foreach(game->opponents, (void (*)(void *))process);
36 }
37
38 void Game_play_step(Game * game) {
39     Ship_actions(game->player, game);
40
41     void processIA(Ship * ship) {
42         Controler_ia(&(ship->control), ship, game);
43         void subprocess(Ship * subship) {
44             if(subship->shipId == ship->shipId) {
45                 return;
46             }
47             double distance = Position_distance(subship->position,
48             → ship->position);
49             if(distance > 2. * ship->width + fabs(ship->moving.x)) {
50                 return;
51             }
52             if(ship->position.x < subship->position.x) {
53                 ship->control.move_right = 0;
54             } else if(ship->position.x > subship->position.x) {
55                 ship->control.move_left = 0;
56             }
57         };
58         List_foreach(game->opponents, (void (*)(void *))subprocess);
59     };
59     List_foreach(game->opponents, (void (*)(void *))processIA);
60
61     void process(Ship * ship) {
62         Ship_actions(ship, game);
63     };
64     List_foreach(game->opponents, (void (*)(void *))process);
65 }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/List.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_LIST
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_LIST
7
8 #include <stdlib.h>
9
10 typedef struct List * List;
11 struct List {
12     void * value;
13     List next;
14 };
15
16 List List_empty();
17
18 void List_free(List * list);
19
20 List List_alloc(void * value, List next);
21
22 void * List_add(List * list, void * value);
23
24 void * List_remove(List * list, void * value);
25
26 void List_FOREACH(List list, void (*process)(void *));
27
28 void * List_predicate(List list, int (*predicate)(void *));
29
30 #endif

```

```

1  /* +-----+ */
2  /* | FICHIER : src/List.c | */
3  /* +-----+ */
4
5 #include "../include/List.h"
6
7 #include <stdlib.h>
8 #include <assert.h>
9

```

```
10 List List_empty() {
11     return NULL;
12 }
13
14 void List_free(List * list) {
15     if(*list == NULL) {
16         return;
17     }
18     List_free(&((*list)->next));
19     free((*list)->value);
20     free(*list);
21     *list = NULL;
22 }
23
24 List List_alloc(void * value, List next) {
25     List list = NULL;
26     assert(list = (List)malloc(sizeof(struct List)));
27     list->value = value;
28     list->next = next;
29     return list;
30 }
31
32 void * List_add(List * list, void * value) {
33     *list = List_alloc(value, *list);
34     return value;
35 }
36
37 void * List_remove(List * list, void * value) {
38     for(; *list != NULL; list = &((*list)->next)) {
39         if((*list)->value == value) {
40             List current = *list;
41             *list = (*list)->next;
42             free(current);
43             return value;
44         }
45     }
46     return NULL;
47 }
48
49 void List_foreach(List list, void (*process)(void *)) {
50     for(; list != NULL; list = list->next) {
```

```

51     (*process)(list->value);
52 }
53 }
54
55 void * List_predicate(List list, int (*predicate)(void *)) {
56     for(; list != NULL; list = list->next) {
57         if((*predicate)(list->value)) {
58             return list->value;
59         }
60     }
61     return NULL;
62 }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/Position.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_POSITION
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_POSITION
7
8 typedef struct Position Position;
9 struct Position {
10     double x;
11     double y;
12 };
13
14 Position Position_create(double x, double y);
15
16 double Position_distance(Position first, Position second);
17
18 #endif
```

```

1  /* +-----+ */
2  /* | FICHIER : src/Position.c | */
3  /* +-----+ */
4
5 #include "../include/Position.h"
6
7 #include <math.h>
```

```
9 Position Position_create(double x, double y) {
10    return (Position){
11        .x = x,
12        .y = y
13    };
14 }
15
16 double Position_distance(Position first, Position second) {
17     return sqrt((second.x - first.x) * (second.x - first.x) +
18                 (second.y - first.y) * (second.y - first.y));
19 }
```

```
1 /* +-----+ */
2 /* / FICHIER : include/Projectile.h / */
3 /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_PROJECTILE
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_PROJECTILE
7
8 #include "Position.h"
9 #include <SDL/SDL.h>
10
11 struct Game;
12 struct Ship;
13
14 typedef struct Projectile Projectile;
15 struct Projectile {
16     Position position;
17     int projectileId;
18     int used;
19     int (*action)(Projectile *, struct Ship *, struct Game *);
20     void (*draw)(Projectile *, SDL_Surface *, int, int, int, int);
21     void * userData;
22 };
23
24 Projectile * Projectile_alloc(Position position);
25
26 void Projectile_free(Projectile * projectile);
27
28 #endif
```

```

1  /* +-----+ */
2  /* | FICHIER : src/Projectile.c | */
3  /* +-----+ */
4
5 #include "../include/Projectile.h"
6
7 #include <stdlib.h>
8 #include <assert.h>
9
10 static int LastProjectileId = 0;
11
12 Projectile * Projectile_alloc(Position position) {
13     Projectile * projectile = NULL;
14     assert(projectile = (Projectile *)malloc(sizeof(Projectile)));
15     projectile->position = position;
16     projectile->projectId = ++LastProjectileId;
17     projectile->used = 0;
18     projectile->action = NULL;
19     projectile->draw = NULL;
20     projectile->userData = NULL;
21     return projectile;
22 }
23
24 void Projectile_free(Projectile * projectile) {
25     if(projectile == NULL) {
26         return;
27     }
28     free(projectile->userData);
29     free(projectile);
30 }
```

```

1  /* +-----+ */
2  /* | FICHIER : include/Ship.h | */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_SHIP
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_SHIP
7
8 #include "Stats.h"
9 #include "Position.h"
```

```
10 #include "List.h"
11 #include "Component.h"
12
13 #define SHIP_SCALE 8
14
15 typedef struct Controler Controler;
16 struct Controler {
17     int move_forward;
18     int move_backward;
19     int move_left;
20     int move_right;
21     int ship_action;
22     int reload_action;
23 };
24
25 struct Game;
26
27 typedef struct Ship Ship;
28 struct Ship {
29     Stats stats;
30     int width;
31     int height;
32     int team;
33     double health;
34     int shipId;
35     Position position;
36     Position moving;
37     List components;
38     List projectiles;
39     Controler control;
40     void * userData;
41     void * dlptr;
42 };
43
44 Controler Controler_create();
45
46 void Controler_draw(SDL_Surface * ecran, Controler * control, Ship
47     * ship);
48 void Controler_move(Controler * control, Ship * ship, struct Game
49     * game);
```

```

49
50 void Controler_ia(Controler * control, Ship * ship, struct Game *
→ game);
51
52 Ship * Ship_alloc(int team, Position position);
53
54 void Ship_free(Ship * ship);
55
56 Ship * Ship_load(const char * path, int team, Position position);
57
58 void Ship_compute(Ship * ship);
59
60 void Ship_draw(Ship * ship, SDL_Surface * ecran);
61
62 void Ship_action(Ship * ship, struct Game * game);
63
64 void Ship_actions(Ship * ship, struct Game * game);
65
66 int Ship_attack(Projectile * projectile, double range, Ship *
→ ship);
67
68 void Ship_apply_damages(double damages, Ship * ship);
69
70 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : src/Ship.c | */
3 /* +-----+ */

4
5 #include "../include/Ship.h"

6
7 #include <stdlib.h>
8 #include <assert.h>
9 #include <dlfcn.h>
10 #include <SDL/SDL.h>
11 #include <SDL/SDL_gfxPrimitives.h>
12 #include "../include/Game.h"

13
14 static int LastShipId = 0;
15

```

```
16     Controleur Controleur_create() {
17         Controleur control;
18         control.move_forward = 0;
19         control.move_backward = 0;
20         control.move_left = 0;
21         control.move_right = 0;
22         control.ship_action = 0;
23         control.reload_action = 0;
24         return control;
25     }
26
27     void Controleur_draw(SDL_Surface * ecran, Controleur * control, Ship
28     * ship) {
29         double coeff;
30         if(control->move_forward) {
31             coeff = 1. + -0.1 * ship->moving.y;
32             if(coeff < 0.1) coeff = 0.1;
33             lineRGBA(ecran,
34                 ship->position.x - ship->width,
35                 ship->position.y - ship->height,
36                 ship->position.x + ship->width,
37                 ship->position.y - ship->height,
38                 255, 255, 0, 255);
39             lineRGBA(ecran,
40                 ship->position.x - ship->width / ((coeff < 1.) ? 1. :
41                 coeff),
42                 ship->position.y - coeff * ship->height,
43                 ship->position.x + ship->width / ((coeff < 1.) ? 1. :
44                 coeff),
45                 ship->position.y - coeff * ship->height,
46                 0, 255, 0, 255);
47         }
48         if(control->move_backward) {
49             coeff = 1. + 0.1 * ship->moving.y;
50             if(coeff < 0.1) coeff = 0.1;
51             lineRGBA(ecran,
52                 ship->position.x - ship->width,
53                 ship->position.y + ship->height,
```

```

54     lineRGBAlpha(ecran,
55         ship->position.x - ship->width / ((coeff < 1.) ? 1. :
56             ~ coeff),
57         ship->position.y + coeff * ship->height,
58         ship->position.x + ship->width / ((coeff < 1.) ? 1. :
59             ~ coeff),
60         ship->position.y + coeff * ship->height,
61         0, 255, 0, 255);
62     }
63     if(control->move_left) {
64         coeff = 1. + -0.1 * ship->moving.x;
65         if(coeff < 0.1) coeff = 0.1;
66         lineRGBAlpha(ecran,
67             ship->position.x - ship->width,
68             ship->position.y - ship->height,
69             ship->position.x - ship->width,
70             ship->position.y + ship->height,
71             255, 255, 0, 255);
72         lineRGBAlpha(ecran,
73             ship->position.x - coeff * ship->width,
74             ship->position.y - ship->height / ((coeff < 1.) ? 1. :
75                 ~ coeff),
76             ship->position.x - coeff * ship->width,
77             ship->position.y + ship->height / ((coeff < 1.) ? 1. :
78                 ~ coeff),
79             0, 255, 0, 255);
80     }
81     if(control->move_right) {
82         coeff = 1. + 0.1 * ship->moving.x;
83         if(coeff < 0.1) coeff = 0.1;
84         lineRGBAlpha(ecran,
85             ship->position.x + ship->width,
86             ship->position.y - ship->height,
87             ship->position.x + ship->width,
88             ship->position.y + ship->height,
89             255, 255, 0, 255);
90         lineRGBAlpha(ecran,
91             ship->position.x + coeff * ship->width,
92             ship->position.y - ship->height / ((coeff < 1.) ? 1. :
93                 ~ coeff),
94             ship->position.x + coeff * ship->width,
95             0, 255, 0, 255);

```

```
90     ship->position.y + ship->height / ((coeff < 1.) ? 1. :
91         ↪ coeff),
92     0, 255, 0, 255);
93 }
94 }
95 void Controler_move(Controler * control, Ship * ship, Game * game)
96 {
97     double tmp;
98     double bonus = (control->reload_action < 50) ? 1. : 0.0625;
99     if(ship->team > 1) {
100         tmp = ship->stats.forward_force;
101         ship->stats.forward_force = ship->stats.backward_force;
102         ship->stats.backward_force = tmp;
103         tmp = ship->stats.left_force;
104         ship->stats.left_force = ship->stats.right_force;
105         ship->stats.right_force = tmp;
106     }
107     double coeff = 1.;
108     if(control->move_forward && !(control->move_backward)) {
109         coeff = bonus * ((ship->moving.y < 0) ? 1. : 3.);
110         ship->moving.y -= coeff * ship->stats.forward_force /
111             ↪ ship->stats.mass;
112     } else if(control->move_backward && !(control->move_forward)) {
113         coeff = bonus * ((ship->moving.y > 0) ? 1. : 3.);
114         ship->moving.y += coeff * ship->stats.backward_force /
115             ↪ ship->stats.mass;
116     } else if(fabs(ship->moving.y) > 0) {
117         double force = 2. * coeff * ((ship->moving.y < 0) ?
118             ↪ (ship->stats.backward_force / ship->stats.mass) :
119             ↪ -(ship->stats.forward_force / ship->stats.mass));
120         if(fabs(ship->moving.y) > fabs(force)) {
121             ship->moving.y += force;
122         } else {
123             ship->moving.y = 0;
124         }
125     }
126     if(control->move_left && !(control->move_right)) {
127         coeff = bonus * ((ship->moving.x < 0) ? 1. : 3.);
128         ship->moving.x -= coeff * ship->stats.left_force /
129             ↪ ship->stats.mass;
```

```

125 } else if(control->move_right && ! (control->move_left)) {
126     coeff = bonus * ((ship->moving.x > 0) ? 1. : 3.);
127     ship->moving.x += coeff * ship->stats.right_force /
128         → ship->stats.mass;
129 } else if(fabs(ship->moving.x) > 0) {
130     double force = 2. * coeff * ((ship->moving.x < 0) ?
131         → (ship->stats.right_force / ship->stats.mass) :
132         → -(ship->stats.left_force / ship->stats.mass));
133     if(fabs(ship->moving.x) > fabs(force)) {
134         ship->moving.x += force;
135     } else {
136         ship->moving.x = 0;
137     }
138 }
139
140 if(ship->position.x < 0) {
141     ship->moving.x = 0;
142     ship->moving.x += 5000. / ship->stats.mass;
143 } else if(ship->position.x > game->width) {
144     ship->moving.x = 0;
145     ship->moving.x += -5000. / ship->stats.mass;
146 }
147
148 if(ship->position.y < 0) {
149     ship->moving.y = 0;
150     ship->moving.y += 5000. / ship->stats.mass;
151 } else if(ship->position.y > game->height) {
152     ship->moving.y = 0;
153     ship->moving.y += -5000. / ship->stats.mass;
154 }
155
156 ship->position.x += ship->moving.x;
157 ship->position.y += ship->moving.y;
158
159 if(control->ship_action && control->reload_action <= 0) {
160     Ship_action(ship, game);
161     control->ship_action = 1;
162     control->reload_action = 20;
163 }
164 --(control->reload_action);
165 if(ship->team > 1) {
166 }
```

```
163     tmp = ship->stats.forward_force;
164     ship->stats.forward_force = ship->stats.backward_force;
165     ship->stats.backward_force = tmp;
166     tmp = ship->stats.left_force;
167     ship->stats.left_force = ship->stats.right_force;
168     ship->stats.right_force = tmp;
169 }
170 }
171
172 void Controler_ia(Controler * control, Ship * ship, Game * game) {
173     if(game->player->position.x < ship->position.x) {
174         control->move_left = 1;
175     } else {
176         control->move_left = 0;
177     }
178     if(game->player->position.x > ship->position.x) {
179         control->move_right = 1;
180     } else {
181         control->move_right = 0;
182     }
183     if(fabs(game->player->position.x - ship->position.x) < 300.) {
184         control->ship_action = 1;
185     } else {
186         control->ship_action = 0;
187     }
188 }
189
190 Ship * Ship_alloc(int team, Position position) {
191     Ship * ship = NULL;
192     assert(ship = (Ship *)malloc(sizeof(Ship)));
193     ship->stats = Stats_zero();
194     ship->width = 0;
195     ship->height = 0;
196     ship->team = team;
197     ship->health = 0.;
198     ship->shipId = ++LastShipId;
199     ship->position = position;
200     ship->moving = Position_create(0., 0.);
201     ship->components = List_empty();
202     ship->projectiles = List_empty();
203     ship->control = Controler_create();
```

```

204     ship->userData = NULL;
205     ship->dptr = NULL;
206     return ship;
207 }
208
209 void Ship_free(Ship * ship) {
210     if(ship == NULL) {
211         return;
212     }
213     if(ship->dptr) {
214         dlclose(ship->dptr);
215     }
216     List_free(&(ship->projectiles));
217     List_free(&(ship->components));
218     free(ship->userData);
219     free(ship);
220 }
221
222 Ship * Ship_load(const char * path, int team, Position position) {
223     void * dptr = NULL;
224     if((dptr = dlopen(path, RTLD_NOW)) == NULL) {
225         fprintf(stderr, "Erreur d'ouverture du vaisseau \"%s\"\n%s\n",
226                 path, dlerror());
227         exit(EXIT_FAILURE);
228     }
229     Ship * (*instantiate)(int, Position) = NULL;
230     if((instantiate = (Ship * (*)(int, Position))dlsym(dptr,
231                 "Ship_instantiate")) == NULL) {
232         fprintf(stderr, "Erreur de chargement du vaisseau
233                 \"%s\"\n%s\n", path, dlerror());
234         exit(EXIT_FAILURE);
235     }
236     Ship * ship = (*instantiate)(team, position);
237     ship->dptr = dptr;
238     Ship_compute(ship);
239     Stats_display(stdout, &(ship->stats));
240     return ship;
241 }
242
243 void Ship_compute(Ship * ship) {
244     ship->stats = Stats_zero();

```

```
242     ship->width = 0;
243     ship->height = 0;
244     void process(Component * component) {
245         Stats_add(&(ship->stats), &(component->stats));
246         int offset = ((ship->team > 1) ? -1 : 1);
247         if(offset * component->position.x + component->width >
248             → ship->width / SHIP_SCALE) {
249             ship->width = offset * component->position.x * SHIP_SCALE +
250             → component->width * SHIP_SCALE;
251         } else if(offset * component->position.x - component->width <
252             → -(ship->width) / SHIP_SCALE) {
253             ship->width = -(offset * component->position.x) * SHIP_SCALE +
254             → + component->width * SHIP_SCALE;
255         }
256         if(offset * component->position.y + component->height >
257             → ship->height / SHIP_SCALE) {
258             ship->height = offset * component->position.y * SHIP_SCALE +
259             → component->height * SHIP_SCALE;
260         } else if(offset * component->position.y - component->height <
261             → -(ship->height) / SHIP_SCALE) {
262             ship->height = -(offset * component->position.y) *
263             → SHIP_SCALE + component->height * SHIP_SCALE;
264         }
265     };
266     List_FOREACH(ship->components, (void (*)(void *))process);
267     ship->health = ship->stats.health;
268 }
```



```
269 void Ship_draw(Ship * ship, SDL_Surface * ecran) {
270     int offset = ((ship->team > 1) ? -1 : 1);
271     /*rectangleRGBAlpha(ecran, ship->position.x - ship->width,
272      → ship->position.y - ship->height, ship->position.x +
273      → ship->width, ship->position.y + ship->height, 255, 0, 0,
274      → 255);*/
275     void process(Component * component) {
276         component->draw(component, ecran, ship->position.x + offset *
277             → SHIP_SCALE * component->position.x, ship->position.y +
278             → offset * SHIP_SCALE * component->position.y, SHIP_SCALE,
279             → ship->team);
280     };
281     List_FOREACH(ship->components, (void (*)(void *))process);
```

```

269 void processProjectiles(Projectile * projectile) {
270     projectile->draw(projectile, ecran, projectile->position.x,
271     → projectile->position.y, SHIP_SCALE, ship->team);
272 };
273 List_FOREACH(ship->projectiles, (void (*)(void
274     → *))processProjectiles);
275 Controler_draw(ecran, &(ship->control), ship);
276 boxRGB(A(ecran, ship->position.x - ship->width - 1,
277     → ship->position.y - ship->height - 1, ship->position.x +
278     → ship->width + 1, ship->position.y - ship->height -
279     → SHIP_SCALE + 1, 0, 0, 0, 255);
280 if(ship->health > 0.) {
281     boxRGB(A(ecran, ship->position.x - ship->width,
282     → ship->position.y - ship->height, ship->position.x +
283     → (ship->health / ship->stats.health) * 2 * ship->width -
284     → ship->width, ship->position.y - ship->height - SHIP_SCALE,
285     → 204, 204, 204, 255);
286 }
287 }
288
289 void Ship_action(Ship * ship, Game * game) {
290     if(ship->health < 0.) {
291         return;
292     }
293     void process(Component * component) {
294         if(component->action)
295             component->action(component, ship, game);
296     };
297     List_FOREACH(ship->components, (void (*)(void *))process);
298 }
299
300 void Ship_actions(Ship * ship, Game * game) {
301     List * current = &(ship->projectiles);
302     Projectile * projectile;
303     while(*current != NULL) {
304         projectile = (Projectile *)((*current)->value);
305         if(! (projectile->action(projectile, ship, game))) {
306             Projectile_free(projectile);
307             List node = *current;
308             *current = (*current)->next;
309             free(node);
310         }
311     }
312 }
```

```
301     } else {
302         current = &((*current)->next);
303     }
304 }
305 ship->health += 0.01 * ship->stats.repair;
306 if(ship->health > ship->stats.health) {
307     ship->health = ship->stats.health;
308 }
309 if(ship->health < 0.) {
310     ship->health = 0.;
311     ship->control.reload_action = 100;
312     return;
313 }
314 Controler_move(&(ship->control), ship, game);
315 }

316 int Ship_attack(Projectile * projectile, double range, Ship *
317 → ship) {
318     int collision = 0;
319     range *= SHIP_SCALE;
320     void processProjectiles(Projectile * current) {
321         if(! collision
322             && Position_distance(projectile->position, current->position)
323             → < 2 * range) {
324             projectile->used = 1;
325             current->used = 1;
326             collision = 1;
327         };
328     List_FOREACH(ship->projectiles, (void (*) (void
329 → *))processProjectiles);
330     if(collision) {
331         return 0;
332     }
333     collision = (projectile->position.x + range > ship->position.x -
334 → ship->width)
335     && (projectile->position.y + range > ship->position.y -
336 → ship->height)
337     && (projectile->position.x - range < ship->position.x +
338 → ship->width)
339     && (projectile->position.y - range < ship->position.y +
340 → ship->height);
```

```

336     if(! collision) {
337         return 0;
338     }
339     projectile->used = 1;
340     return 1;
341 }
342
343 void Ship_apply_damages(double damages, Ship * ship) {
344     ship->health -= 200. * damages / ship->stats.armor;
345 }
```

```

1  /* +-----+ */
2  /* / FICHIER : include/Stats.h / */
3  /* +-----+ */
4
5 #ifndef DEF_HEADER_ESGI_SPACE_CRAFT_STATS
6 #define DEF_HEADER_ESGI_SPACE_CRAFT_STATS
7
8 #include <stdio.h>
9
10 typedef struct Stats Stats;
11 struct Stats {
12     double forward_force;
13     double backward_force;
14     double right_force;
15     double left_force;
16     double mass;
17     double health;
18     double armor;
19     double damage;
20     double repair;
21     int points;
22 };
23
24 Stats Stats_zero();
25
26 Stats Stats_base(double mass, double health, double armor, int
27   → points);
28 void Stats_add(Stats * current, const Stats * other);
```

```
29
30 void Stats_display(FILE * flow, const Stats * stats);
31
32 #endif
```



```
1 /* +-----+ */
2 /* / FICHIER : src/Stats.c / */
3 /* +-----+ */

4
5 #include "../include/Stats.h"

6
7 Stats Stats_zero() {
8     return (Stats){
9         .forward_force = 0.,
10        .backward_force = 0.,
11        .right_force = 0.,
12        .left_force = 0.,
13        .mass = 0.,
14        .health = 0.,
15        .armor = 0.,
16        .damage = 0.,
17        .repair = 0.,
18        .points = 0
19    };
20 }
21
22 Stats Stats_base(double mass, double health, double armor, int
23 → points) {
24     Stats stats = Stats_zero();
25     stats.mass = mass;
26     stats.health = health;
27     stats.armor = armor;
28     stats.points = points;
29     return stats;
30 }
31
32 void Stats_add(Stats * current, const Stats * other) {
33     current->forward_force += other->forward_force;
34     current->backward_force += other->backward_force;
35     current->right_force += other->right_force;
```

```
35     current->left_force += other->left_force;
36     current->mass += other->mass;
37     current->health += other->health;
38     current->armor += other->armor;
39     current->damage += other->damage;
40     current->repair += other->repair;
41     current->points += other->points;
42 }
43
44 void Stats_display(FILE * flow, const Stats * stats) {
45     fprintf(flow, "Stats : {\n"
46         " - forward force : %g\n"
47         " - backward force : %g\n"
48         " - left force : %g\n"
49         " - right force : %g\n"
50         " - mass : %g\n"
51         " - health : %g\n"
52         " - armor : %g\n"
53         " - damage : %g\n"
54         " - repair : %g\n"
55         " - points : %d\n"
56     "}\n",
57     stats->forward_force,
58     stats->backward_force,
59     stats->left_force,
60     stats->right_force,
61     stats->mass,
62     stats->health,
63     stats->armor,
64     stats->damage,
65     stats->repair,
66     stats->points);
67 }
```

A.14 Opérations bit-à-bit

A.14.1 Entraînement

Correction 128 (★★ Application opérations bit-à-bit).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     unsigned char entier8;
6     unsigned int entier32;
7     unsigned int entier32bis;
8     unsigned long entier64;
9     int test;
10
11    entier32 = 1 << 14;
12    printf("%08x (15e bit à 1)\n", entier32);
13    entier32 = ~((unsigned int)1 << 13);
14    printf("%08x (14e bit à 0)\n", entier32);
15
16    entier64 = (unsigned long)1 << 42;
17    printf("%016lx (43e bit à 1)\n", entier64);
18    entier64 = ~((unsigned long)1 << 2);
19    printf("%016lx (3e bit à 0)\n", entier64);
20
21    entier32 = (unsigned int)1 << 12 | (unsigned int)1 << 11;
22    printf("%08x (13e et 12e bit à 1)\n", entier32);
23
24    entier32 &= ~((unsigned int)1 << 11);
25    printf("%08x (12e bit à 0)\n", entier32);
26
27    entier32 ^= (unsigned int)1 << 12;
28    printf("%08x (changement 13e bit)\n", entier32);
29
30    entier8 = 0x1;
31    entier32 &= ~((unsigned int)1 << 10);
32    entier32 |= (unsigned int)entier8 << 10;
33    printf("%08x (affectation 11e bit)\n", entier32);
34
35    test = (entier32 & ((unsigned int)1 << 10)) != 0;
```

```

36 printf("%08x (test 11e bit : %d)\n", entier32, test);

37
38 entier32 = 0xFF;
39 test = (entier32 & 0x7) == 0x7;
40 printf("%08x (test des 3 bits de poids faible à 1 : %d)\n",
41     entier32, test);

42
43 entier32 = 0xF0;
44 test = (entier32 & 0xF) == 0xF;
45 printf("%08x (test des 4 bits de poids faible à 0 : %d)\n",
46     entier32, test);

47
48 entier32 = 0xFFE80F12;
49 entier32bis = 0x0017FOED;

50
51 test = (entier32 ^ (~entier32bis)) == 0;
52 printf("%08x et %08x (test bits tous différents : %d)\n",
53     entier32, entier32bis, test);

54
55 exit(EXIT_SUCCESS);
56 }
```

Correction 129 (★★★ Gérer une grille sur un entier).

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef unsigned long Bit8x8Grid;
5
6 Bit8x8Grid Bit8x8Grid_creer();
7
8 int Bit8x8Grid_getoffset(int x, int y);
9
10 void Bit8x8Grid_afficher(FILE * flow, Bit8x8Grid grille);
11
12 void Bit8x8Grid_set(Bit8x8Grid * grille, int x, int y, unsigned
13     ↪ char valeur);
14
15 unsigned char Bit8x8Grid_get(Bit8x8Grid grille, int x, int y);
```

```
16 int main() {
17     Bit8x8Grid grille = Bit8x8Grid_creer();
18     Bit8x8Grid_set(&grille, 3, 1, 1);
19     Bit8x8Grid_set(&grille, 7, 7, 1);
20     Bit8x8Grid_afficher(stdout, grille);
21     exit(EXIT_SUCCESS);
22 }
23
24 Bit8x8Grid Bit8x8Grid_creer() {
25     return (Bit8x8Grid)0;
26 }
27
28 int Bit8x8Grid_getoffset(int x, int y) {
29     return x + 8 * y;
30 }
31
32 void Bit8x8Grid_afficher(FILE * flow, Bit8x8Grid grille) {
33     int x, y;
34     for(y = 0; y < 8; ++y) {
35         for(x = 0; x < 8; ++x) {
36             fputc(Bit8x8Grid_get(grille, x, y) ? '#' : '~', flow);
37         }
38         fputc('\n', flow);
39     }
40 }
41
42 void Bit8x8Grid_set(Bit8x8Grid * grille, int x, int y, unsigned
43 →     char valeur) {
44     int offset = Bit8x8Grid_getoffset(x, y);
45     valeur = !!valeur;
46     *grille &= ~((Bit8x8Grid)1 << offset);
47     *grille |= (Bit8x8Grid)valeur << offset;
48 }
49
50 unsigned char Bit8x8Grid_get(Bit8x8Grid grille, int x, int y) {
51     int offset = Bit8x8Grid_getoffset(x, y);
52     return !(grille & ((Bit8x8Grid)1 << offset));
```

Correction 130 (★★★ Buffer pour lecture et écriture bit-à-bit).

```

1 CC= gcc
2 CFLAGS= -O2 -Wall -Wextra -Werror -ansi
3 CLIBS= -lm
4 DEFS=
5 EXE= executable
6 OBJ= main.o \
    bitbuffer.o
8
9 $(EXE) : $(OBJ)
10   $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)
11
12 main.o : main.c bitbuffer.h
13 tree.o : bitbuffer.c bitbuffer.h
14
15 %.o : %.c
16   $(CC) $(CFLAGS) -c $< $(DEFS)
17
18 clean :
19   rm -rf $(OBJ)
20   rm -rf $(EXE)

```

```

1 /* +-----+ */
2 /* / FICHIER : main.c / */
3 /* +-----+ */
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "bitbuffer.h"
8
9 int main() {
10     BitBuffer buffer = BitBuffer_creer();
11     unsigned long infos[] = {
12         1, 0x1,
13         5, 0xf,
14         16, 0xffff,
15         32, 0x18181818,
16         5, 0x7,
17         16, 0xffff,
18         32, 0x18181818,
19         5, 0xf,

```

```
20     1, 0x1,
21     1, 0x1,
22     1, 0x0,
23     1, 0x1,
24     1, 0x0,
25     1, 0x0,
26     1, 0x1,
27     16, 0xffff,
28     31, 0x8181818,
29     5, 0x7,
30     31, 0x8181818,
31     5, 0xf,
32     0
33 };
34 int i;
35 for(i = 0; infos[2 * i] > 0; ++i) {
36     BitBuffer_write(&buffer, infos[2 * i + 1], infos[2 * i]);
37 #ifdef VERBOSE
38     BitBuffer_print(stderr, &buffer);
39 #endif
40 }
41 unsigned long data = 0;
42 for(i = 0; infos[2 * i] > 0; ++i) {
43     BitBuffer_read(&buffer, &data, infos[2 * i]);
44 #ifdef VERBOSE
45     BitBuffer_print(stderr, &buffer);
46 #endif
47     printf("(d) %lx\n", data == infos[2 * i + 1], data);
48 }
49 BitBuffer_free(&buffer);
50 exit(EXIT_SUCCESS);
51 }
```

```
1 /* +-----+ */
2 /* / FICHIER : bitbuffer.h / */
3 /* +-----+ */
4 #ifndef DEF_HEADER_BITBUFFER
5 #define DEF_HEADER_BITBUFFER
6
7 #include <stdio.h>
```

```

8
9 typedef struct BitBuffer BitBuffer;
10 struct BitBuffer {
11     unsigned long * data;
12     long cursor;
13     long capacite;
14 };
15
16 BitBuffer BitBuffer_creer();
17
18 int BitBuffer_update(BitBuffer * buffer, unsigned int
→ adding_bits);
19
20 int BitBuffer_write(BitBuffer * buffer, unsigned long data,
→ unsigned int bits);
21
22 int BitBuffer_read(BitBuffer * buffer, unsigned long * data,
→ unsigned int bits);
23
24 void BitBuffer_print(FILE * flow, const BitBuffer * buffer);
25
26 void BitBuffer_free(BitBuffer * buffer);
27
28 #endif

```

```

1 /* +-----+ */
2 /* | FICHIER : bitbuffer.c | */
3 /* +-----+ */
4 #include "bitbuffer.h"

5
6 #include <stdlib.h>

7
8 BitBuffer BitBuffer_creer() {
9     BitBuffer buffer;
10    buffer.data = NULL;
11    buffer.cursor = 0;
12    buffer.capacite = 0;
13    return buffer;
14 }
15

```

```
16 int BitBuffer_update(BitBuffer * buffer, unsigned int adding_bits)
17 {  
18     int size = (buffer->cursor + adding_bits + 8 * sizeof(unsigned  
19         long)) / (8 * sizeof(unsigned long));  
20     if(size < buffer->capacite) {  
21         return 1;  
22     }  
23     int newCapacite = size * 2 + 10;  
24     unsigned long * tmp = NULL;  
25     if((tmp = (unsigned long *)realloc(buffer->data, newCapacite *  
26         sizeof(unsigned long))) == NULL) {  
27         return 0;  
28     }  
29     buffer->data = tmp;  
30     buffer->capacite = newCapacite;  
31     return 1;  
32 }  
33  
34 int BitBuffer_write(BitBuffer * buffer, unsigned long data,  
35     unsigned int bits) {  
36     unsigned long mask = 0xffffffffffffffff;  
37     if(! BitBuffer_update(buffer, bits)) {  
38         return 0;  
39     }  
40     unsigned long offset = buffer->cursor / (8 * sizeof(unsigned  
41         long));  
42     unsigned long bitoffset = buffer->cursor % (8 * sizeof(unsigned  
43         long));  
44     buffer->data[offset] &= ~(mask << bitoffset);  
45     buffer->data[offset] |= data << bitoffset;  
46     if(bitoffset + bits >= (8 * sizeof(unsigned long))) {  
47         ++offset;  
48         buffer->data[offset] &= ~(mask >> (8 * sizeof(unsigned long) -  
49             bitoffset));  
50         buffer->data[offset] |= data >> (8 * sizeof(unsigned long) -  
51             bitoffset);  
52     }  
53     buffer->cursor += bits;  
54     return 1;  
55 }
```

```

49 int BitBuffer_read(BitBuffer * buffer, unsigned long * data,
50     unsigned int bits) {
51     if(bits > buffer->cursor) {
52         return 0;
53     }
54     unsigned long mask = 0xffffffffffffffff;
55     unsigned long offset = buffer->cursor / (8 * sizeof(unsigned
56     long));
57     *data = buffer->data[0] & (mask >> (8 * sizeof(unsigned long) -
58     bits)));
59     unsigned long i;
60     for(i = 0; i <= offset; ++i) {
61         buffer->data[i] >>= bits;
62         if(i + 1 > offset) {
63             continue;
64         }
65         buffer->data[i] |= buffer->data[i + 1] << (8 * sizeof(unsigned
66         long) - bits);
67     }
68     buffer->cursor -= bits;
69     return 1;
70 }

71 void BitBuffer_print(FILE * flow, const BitBuffer * buffer) {
72     long c = buffer->cursor;
73     for(--c; c >= 0; --c) {
74         fputc((buffer->data[c / (8 * sizeof(unsigned long))]] &
75             ((unsigned long)1 << (c % (8 * sizeof(unsigned long))))) ?
76             '1' : '0', flow);
77     }
78     fputc('\n', flow);
79 }

80 void BitBuffer_free(BitBuffer * buffer) {
81     if(buffer->data == NULL) {
82         free(buffer->data);
83         buffer->data = NULL;
84     }
85     buffer->cursor = 0;
86     buffer->capacite = 0;
87 }
```

B Examens des années précédentes

B.1 Données statistiques partiels 2021 - 2023

Pour vous permettre d'appréhender au mieux les examens terminaux des semestres 1 et 2, vous retrouverez ici des informations sur leur déroulé les années précédentes :

- Des statistiques sur les résultats obtenus par les candidats sur l'année précédente.
- Les sujets sur lesquels les candidats ont été évalué pour vous tester en amont de l'examen.

Vous pouvez vous référer aux conseils promulgués pour réussir ces examens en section **0.3**.

Les données suivantes représentent les notes obtenues aux examens finaux correspondants aux sujets donnés en section **B.2** et **B.3**. À noter que le sujet zéro servira d'entraînement lors de l'examen blanc (dans le cas où vous voudriez vous tester en situation réelle sans vous spoiler).

B.1.1 Semestre 1

Alternance

Session :	2021-22	2022-23	2023-24
Effectif :	91	126	99
Moyenne :	14.8	9.8	9.9
Écart type :	4.6	3.6	3.9
Minimale :	0	1	2
Premier quartile :	12	8	7
Médiane :	16	10	9
Troisième quartile :	18	13	12
Maximale :	20	20	20

Janvier

Session :	2021-22	2022-23	2023-24 i	2023-24 a
Effectif :	38	26	30	27
Moyenne :	13.6	10.5	9.1	8.6
Écart type :	4.5	3	3.6	4.9
Minimale :	3	4	4	1
Premier quartile :	10	8	7	4
Médiane :	14	11	8	8
Troisième quartile :	17	13	11	12
Maximale :	20	15	18	19

B.1.2 Semestre 2

Alternance

Session :	2021-22	2022-23	2023-24
Effectif :	72	117	91
Moyenne :	9.8	8.6	8.3
Écart type :	4.8	4	4
Minimale :	0	1	2
Premier quartile :	6	6	5
Médiane :	10	9	8
Troisième quartile :	13	11	11
Maximale :	20	19	19

Janvier

Session :	2021-22	2022-23	2023-24
Effectif :	36	24	28
Moyenne :	7	9.2	7.6
Écart type :	4.4	4.4	2.5
Minimale :	0	2	3
Premier quartile :	4	6	6
Médiane :	7	9	8
Troisième quartile :	9	12	9
Maximale :	17	18	13

Les sujets donnés ici sont des exemples de sujets tombés à l'examen de fin de semestre les années précédentes pour les formations en alternance et / ou la formation en initial entrée en Janvier.

B.2 Sujets partiels semestre 1

Lors du semestre 1, les étudiants étaient évalués sur les notions des sections Variables, Expression, Conditions et Boucles (sauf en 2021-2022).

B.2.1 S1 Sujet zéro

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie de faire saisir un entier à l'utilisateur puis de l'afficher.

Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
<?Langage C

Lire "Entrez un mot de passe : ", mot de passe

Si mot de passe === 1235 Alors

Afficher "Bienvenue !", fin ligne

Sinon

Afficher "Au revoir !", fin ligne
quitter

Fin Si

?>
```

Exercice 2 (..... / 4 points)

Objectif : Vérifier les bornes supérieures des entiers non signés.

Un camarade a trouvé sur internet les valeurs maximales en hexadécimal et décimal d'entiers et aimerait vérifier si c'est vrai en le codant par lui-même. Internet lui dit que pour les types d'entiers allant jusqu'à 8 octets, il trouverait les valeurs suivantes :

```
taille      hexadecimal      (decimal)
8 octets : ffffffffffffffff (18446744073709551615)
4 octets : 00000000ffff (        4294967295)
2 octets : 000000000000ffff (        65535)
1 octets : 00000000000000ff (        255)
```

Ce camarade a essayé le code suivant. Proposer une version fonctionnelle de ce code en langage C donnant exactement la même sortie que celle donnée :

```
entier8octets big <- ffffffffffffff
entier4octets not_big <- big
entier2octets not_so_big <- big
entier1octet not_big_at_all <- big
Afficher "taille      hexadecimal      (decimal)", fin ligne
Afficher "8 octets : ", hexa big, big, fin ligne
Afficher "4 octets : ", hexa not_big, not_big, fin ligne
Afficher "2 octets : ", hexa not_so_big, not_so_big, fin ligne
Afficher "1 octets : ", hexa not_big_at_all, not_big_at_all, fin
↪  ligne
```

Exercice 3 (..... / 4 points)

Objectif : calculatrice réelle.

- **Entrée 1** : un réel (nombre 1)
- **Entrée 2** : un caractère (opérateur)
- **Entrée 3** : un réel (nombre 2)

Affiche le résultat du calcul donné par les deux nombres et l'opérateur.

Sortie 1 :
Addition simple.

```
>>> 51.25 + 42.75  
51.25 + 42.75 = 94
```

Sortie 2 :
Soustraction écriture scientifique.

```
>>> 5e+100 - 3e+99  
5e+100 - 3e+99 = 4.7e+100
```

Sortie 3 :
Multiplication.

```
>>> 5.125 * 0.125  
5.125 * 0.125 = 0.640625
```

Sortie 4 :
Division fractionnaire.

```
>>> 5 / 2.1  
5 / 2.1 = 2.38095
```

Sortie 5 :
Modulo à virgule.

```
>>> 5 % 2.1  
5 % 2.1 = 0.8
```

Exercice 4 (..... / 4 points)

Objectif : Appliquer un chiffre de César à une suite de caractères.

- **Entrée 1** : un entier (clé).
- **Entrée 2** : une suite de caractères (message).
- **Entrée 3** : un retour à la ligne (marqueur de fin).

Un chiffre de César est un décalage d'une clé donnée appliqué dans un alphabet. Ceci est cyclique, par conséquent :

- Le caractère qui suit 'z' est 'a'.
- Le caractère qui suit 'Z' est 'A'.
- Le caractère qui suit '9' est '0'.
- Un autre caractère est laissé inchangé.

Exemple de sortie :

Codage simple clé 5 :

```
Entrez une clé, puis un message : 5 Exemple de texte
Jcjruqj ij yjcyj
```

Exemple de sortie :

Décodage simple clé 5 :

```
Entrez une clé, puis un message : -5 Jcjruqj ij yjcyj
Exemple de texte
```

Exemple de sortie :

Codage long clé 42 :

```
Entrez une clé, puis un message : 42 Le Langage C vous apportera de nombreuses
→  reponses, en particulier pour expliciter la magie de Python !
Bu Bqdwaqw S leki qffehjuhq tu decrhukiui hufediui, ud fqhjyskbyuh fekh
→  unfbysyjuh bq cqwyu tu Fojxed !
```

Exercice 5 (..... / 4 points)

Objectif : Décoder un nombre en base quelconque.

- **Entrée 1** : un entier sur 8 octets (nombre)
- **Entrée 2** : un entier sur 4 octets (base)

Lire un entier sur 8 octets. Cette entier devra être converti dans la base donnée. Les chiffres de ce nombre dans la base b entière donnée sont représentés de la manière suivante :

- Chiffres de 0 à 9 pour les chiffres de 0 à 9.
- Lettres de **a** à **z** (minuscules et majuscules acceptées) pour les chiffres de 10 à 35.
- Format `#{valeur}` pour tout autre chiffre où valeur est un entier positif inférieur à b .

Pour rappel, un nombre à n chiffres exprimé dans une base b par les chiffres $(c_i)_{i \in [n]}$ comme $(c_1 c_2 c_3 \cdots c_n)_b$ se calcule comme il suit :

$$c_1 b^{n-1} + c_2 b^{n-2} + \cdots + c_{n-1} b^1 + \cdots + c_n b^0$$

Exemple de sortie :

Base binaire.

```
Entrez une valeur en base décimale et la base souhaitée : 42 2
(42)_10 = (101010)_2
```

Exemple de sortie :

Base hexadécimale, grande valeur.

```
Entrez une valeur en base décimale et la base souhaitée : 18446744073709551615 16
(18446744073709551615)_10 = (fffffffffffff_16
```

Exemple de sortie :

Base 100, avec chiffre non alphanumérique.

```
Entrez une valeur en base décimale et la base souhaitée : 44202 100
(44202)_10 = (4#{42}2)_100
```

Exemple de sortie :
Base 42, codage de texte court.

```
Entrez une valeur en base décimale et la base souhaitée : 81204493748 42
(81204493748)_10 = (exemple)_42
```

B.2.2 S1 2021-2022 : Alternance

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5	Exercice 6
..... / 3 / 3 / 3 / 4 / 5 / 2

Modalités

- L'étudiant a 2 heures pour composer sur ce sujet et rendre une copie papier de sa production.
- Les 6 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en langage C tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 4 (structures conditionnelles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- Le barème utilisé vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.

Bon courage !

Exercice 1 (..... / 3 points)

Objectif : adapter un code en langage C.

Un élève débutant en langage C essaie d'affecter 42 à un entier puis de l'afficher. Le compilateur refuse de compiler son code. Adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
include <<studio>>
main :
    entier = 42
    print(entier)
```

Exercice 2 (..... / 3 points)

Objectif : vérifier un mot de passe.

- Entrée 1 : un entier (nombre)

Lire un entier, selon sa valeur :

- S'il vaut 1235, afficher un accès autorisé.
- Sinon, afficher un accès refusé.

Exemple de sortie :

Accès autorisé.

```
Entrez un mot de passe : 1235
Accès autorisé !
```

Exemple de sortie :

Accès refusé.

```
Entrez un mot de passe : 0420
Accès refusé !
```

Exercice 3 (..... / 3 points)

Objectif : lire un entier et visualiser un dépassement de capacité.

- **Entrée 1** : un entier sur 8 octets (nombre)

Lire un entier sur 8 octets puis afficher sa visualisation :

- décimale sur 4 octets.
- décimale sur 8 octets.
- hexadécimale sur 4 octets (sur 8 caractères, comblés par des zéros).
- hexadécimale sur 8 octets (sur 16 caractères, comblés par des zéros).

Exemple de sortie :

Petite valeur.

```
Entrez un entier : 42
Decimal sur 4 octets : 42
Decimal sur 8 octets : 42
Hexadecimal sur 4 octets : 0000002a
Hexadecimal sur 8 octets : 000000000000002a
```

Exemple de sortie :

Négative en 4 octets.

```
Entrez un entier : 4000000000
Decimal sur 4 octets : -294967296
Decimal sur 8 octets : 4000000000
Hexadecimal sur 4 octets : ee6b2800
Hexadecimal sur 8 octets : 00000000ee6b2800
```

Exemple de sortie :

Dépassement de capacité du 8 octets.

```
Entrez un entier : 8000000000000000
Decimal sur 4 octets : 1939144704
Decimal sur 8 octets : 8000000000000000
Hexadecimal sur 4 octets : 73950000
Hexadecimal sur 8 octets : 000048c273950000
```

Exercice 4 (..... / 4 points)

Objectif : déterminer un minimum et un maximum.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un entier (nombre 2)
- **Entrée 3** : un entier (nombre 3)

Lire trois nombres entiers puis :

- Afficher le minimum des trois nombres.
- Afficher le maximum des trois nombres.

Exemple de sortie :

Ordre croissant.

```
Entrez 3 entiers : 1 2 3
min(1, 2, 3) = 1
max(1, 2, 3) = 3
```

Exemple de sortie :

Ordre décroissant.

```
Entrez 3 entiers : 3 2 1
min(3, 2, 1) = 1
max(3, 2, 1) = 3
```

Exemple de sortie :

Ordre quelconque.

```
Entrez 3 entiers : 5 7 2
min(5, 7, 2) = 2
max(5, 7, 2) = 7
```

Exercice 5 (..... / 5 points)

Objectif : calculatrice d'entiers.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un caractère (opérateur)
- **Entrée 3** : un entier (nombre 2)

Lire un nombre, un opérateur et un autre nombre puis afficher le calcul pour les opérations :

- '+' pour l'addition d'entiers.
- '-' pour la soustraction d'entiers.
- '*' pour la multiplication d'entiers.
- '/' pour la division fractionnaire (à virgule).
- 'd' pour la division entière (quotient de la division euclidienne).
- 'm' pour le modulo (reste de la division euclidienne).

Exemple de sortie :

Addition.

```
>>> 5 + 4  
= 9
```

Exemple de sortie :

Multiplication.

```
>>> 100000000 * 10000000  
= 1000000000000000000
```

Exemple de sortie :

Division fractionnaire.

```
>>> 7 / 5  
= 1.4
```

Exemple de sortie :

Division entière.

```
>>> 7 d 5  
= 1
```

Exemple de sortie :

Modulo.

```
>>> 7 m 5  
= 2
```

Exemple de sortie :

Gestion division par zéro.

```
>>> 7 / 0  
= inf
```

Exercice 6 (..... / 2 points)

Objectif : déterminer hauteur et arrivé d'un lancer de projectile.

- **Entrée 1** : un réel (coordonnée position initiale abscisses x)
- **Entrée 2** : un réel (coordonnée position initiale ordonnées y)
- **Entrée 3** : un réel (coordonnée vecteur vitesse initiale abscisses x)
- **Entrée 4** : un réel (coordonnée vecteur vitesse initiale ordonnées y)

La trajectoire d'un projectile à un temps t donné est exprimée en fonction d'une position initiale (P_x, P_y) , d'un vecteur de vitesse initiale (V_x, V_y) et de la valeur de pesanteur $g \simeq 9.81$ par la relation :

$$\begin{cases} x = V_x \cdot t + P_x \\ y = -\frac{1}{2}g \cdot t^2 + V_y \cdot t + P_y \end{cases}$$

Le projectile atteint sa hauteur maximale pour :

$$t_{max} = \frac{V_y}{g}$$

et touche le sol pour :

$$t_{impact} = \frac{V_y + \sqrt{V_y^2 + 2g \cdot P_y}}{g}$$

Dans votre programme, indiquez depuis la position initiale et le vecteur de vitesse initiale :

- La relation qui guide la trajectoire.
- La position et temps où la hauteur maximale est atteinte.
- La position et temps où le projectile touche le sol.

Exemple de sortie :

Depuis la position (1, 2) avec pour vitesse initiale (8, 4).

```
Entrez les coordonnées de départ du projectile : 1 2
Entrez le vecteur vitesse initial : 8 4
Le projectile va suivre la trajectoire en fonction du temps t par :
x = 8 t + 1
y = -4.905 t^2 + 4 t + 2
Arrive à hauteur maximale à t = 0.407747 et coordonnées (4.26198, 2.81549)
Arrive au sol à t = 1.16538 et coordonnées (10.323, 0)
```

B.2.3 S1 2021-2022 : Janvier

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 3 / 3 / 4 / 5 / 5

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 4 (structures conditionnelles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- Le barème utilisé vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 3 points)

Objectif : adapter un code en langage C.

Un élève débutant en langage C essaie de faire saisir un entier à l'utilisateur puis de l'afficher. Le compilateur refuse de compiler son code. Adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
import studio

Sub prog
    read "Entrez un entier : ", entier
    print "Voici un entier : ", entier
End Sub
```

Exercice 2 (..... / 3 points)

Objectif : donner le plus grand nombre.

- **Entrée 1** : un entier (nombre)
- **Entrée 2** : un entier (nombre)

Lire deux entiers, afficher la valeur du plus grand.

Exemple de sortie :
Premier plus grand.

```
Saisir deux entiers : 1337 111
1337 est le plus grand.
```

Exemple de sortie :
Second plus grand.

```
Saisir deux entiers : 42 1235
1235 est le plus grand.
```

Exercice 3 (..... / 4 points)

Objectif : sommer des éléments sous condition.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un entier (nombre 2)
- **Entrée 3** : un entier (nombre 3)

Lire trois nombres entiers puis :

- Afficher la somme des nombres pairs.
- Afficher la somme des nombres impairs.

Exemple de sortie :

Impairs uniquement.

```
Entrez 3 nombres : 1 3 5
Somme des éléments pairs : 0
Somme des éléments impairs : 9
```

Exemple de sortie :

Pairs uniquement.

```
Entrez 3 nombres : 2 4 8
Somme des éléments pairs : 14
Somme des éléments impairs : 0
```

Exemple de sortie :

Pairs et impairs mélangés.

```
Entrez 3 nombres : 7 10 3
Somme des éléments pairs : 10
Somme des éléments impairs : 10
```

Exercice 4 (..... / 5 points)

Objectif : calculatrice d'entiers.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un caractère (opérateur)
- **Entrée 3** : un entier (nombre 2)

Lire un nombre, un opérateur et un autre nombre puis afficher le calcul pour les opérations :

- '+' pour l'addition d'entiers.
- '-' pour la soustraction d'entiers.
- '*' pour la multiplication d'entiers.
- '/' pour la division fractionnaire (à virgule).
- 'd' pour la division entière (quotient de la division euclidienne).
- 'm' pour le modulo (reste de la division euclidienne).

Exemple de sortie :
Addition.

```
>>> 5 + 4  
= 9
```

Exemple de sortie :
Multiplication.

```
>>> 100000000 * 10000000  
= 10000000000000000
```

Exemple de sortie :
Division fractionnaire.

```
>>> 7 / 5  
= 1.4
```

Exemple de sortie :
Division entière.

```
>>> 7 d 5  
= 1
```

Exemple de sortie :
Modulo.

```
>>> 7 m 5  
= 2
```

Exemple de sortie :
Gestion division par zéro.

```
>>> 7 / 0  
= inf
```

Exercice 5 (..... / 5 points)

Objectif : lire un entier et visualiser un dépassement de capacité.

- **Entrée 1** : un entier sur 8 octets (nombre)

Lire un entier sur 8 octets puis afficher sa visualisation :

- hexadécimale et décimale sur 8 octets.
- hexadécimale et décimale sur 4 octets.
- hexadécimale et décimale sur 2 octets.
- hexadécimale et décimale sur 1 octets.
- découpée en hexadécimal (comblée automatiquement de zéros et séparée par des tirets-bas) :
 - 4 premiers octets (octets de poids fort d'un entier sur 8 octets)
 - 2 octets suivants (octets de poids fort d'un entier sur 4 octets)
 - 1 octet suivant (octets de poids fort d'un entier sur 2 octets)
 - 1 octet suivant (octets d'un entier sur 1 octet)

Exemple de sortie :

Petite valeur.

```
Entrez en entier : 42
entier sur 8 octets :          2a (42)
entier sur 4 octets :          2a (42)
entier sur 2 octets :          2a (42)
entier sur 1 octet :          2a (42)
découpe : 00000000_0000_00_2a
```

Exemple de sortie :

Limite sur 4 octets.

```
Entrez en entier : 4000000000
entier sur 8 octets :          ee6b2800 (4000000000)
entier sur 4 octets :          ee6b2800 (4000000000)
entier sur 2 octets :          2800 (10240)
entier sur 1 octet :          0 (0)
découpe : 00000000_ee6b_28_00
```

Exemple de sortie :

Dépassement de capacité du 8 octets.

```
Entrez en entier : 8000000000000000
entier sur 8 octets :      48c273950000 (8000000000000000)
entier sur 4 octets :      73950000 (1939144704)
entier sur 2 octets :      0 (0)
entier sur 1 octet :      0 (0)
découpe : 000048c2_7395_00_00
```

B.2.4 S1 2022-2023 : Alternance

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie de faire saisir un entier à l'utilisateur puis de l'afficher.

Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
import Langage C

Lire "Entrez deux nombres entiers : ", nombre min, nombre max

Si [nombre max] plus petit que [nombre min] :

    echanger [nombre min] et [nombre max]

Fin Si

Afficher "Le plus petit est ", nombre min, " et le plus grand ", nombre
↪ max
```

Exercice 2 (..... / 4 points)

Objectif : calculatrice d'entiers.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un caractère (opérateur)
- **Entrée 3** : un entier (nombre 2)

Lire un nombre, un opérateur et un autre nombre puis afficher le calcul pour les opérations :

- '+' pour l'addition d'entiers.
- '-' pour la soustraction d'entiers.
- '*' pour la multiplication d'entiers.
- '/' pour la division fractionnaire (à virgule).
- 'd' pour la division entière (quotient de la division euclidienne).
- 'm' pour le modulo (reste de la division euclidienne).

Exemple de sortie :

Addition.

```
>>> 5 + 4  
= 9
```

Exemple de sortie :

Multiplication.

```
>>> 100000000 * 10000000  
= 1000000000000000000
```

Exemple de sortie :

Division fractionnaire.

```
>>> 7 / 5  
= 1.4
```

Exemple de sortie :

Division entière.

```
>>> 7 d 5  
= 1
```

Exemple de sortie :

Modulo.

```
>>> 7 m 5  
= 2
```

Exemple de sortie :

Gestion division par zéro.

```
>>> 7 / 0  
= inf
```

Exercice 3 (..... / 4 points)

Objectif : Déterminer une moyenne.

- **Entrée 1** : séquence de notes séparées par des espacements.
- **Entrée 2** : valeur négative (marqueur de fin)

Lire une suite de taille non déterminée de notes. Puis en calculer la moyenne.
Une valeur négative marque la fin de la séquence.

Rappel : une moyenne de valeurs $(x_i)_{i \in [n]}$ se calcule de la manière suivante :

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

Exemple de sortie :

Moyenne de notes (résultat entier) :

```
Entrez de notes (négative pour terminer) : 10 12 14 16 18 -1
La moyenne de ces notes est : 14
```

Exemple de sortie :

Moyenne de notes (résultat à virgule) :

```
Entrez de notes : 10 11 13 -1
La moyenne de ces notes est : 11.3333
```

Exemple de sortie :

Aucunes notes :

```
Entrez de notes (négative pour terminer) : -1
Pas assez de notes saisies.
```

Exercice 4 (..... / 4 points)

Objectif : lire un entier et visualiser un dépassement de capacité.

- **Entrée 1** : un entier sur 8 octets (nombre)

Lire un entier sur 8 octets puis afficher sa visualisation :

- hexadécimale et décimale sur 8 octets.
- hexadécimale et décimale sur 4 octets.
- hexadécimale et décimale sur 2 octets.
- hexadécimale et décimale sur 1 octet.
- découpée en hexadécimal (comblée automatiquement de zéros et séparée par des tirets-bas) :
 - 4 premiers octets (octets de poids fort d'un entier sur 8 octets)
 - 2 octets suivants (octets de poids fort d'un entier sur 4 octets)
 - 1 octet suivant (octets de poids fort d'un entier sur 2 octets)
 - 1 octet suivant (octets d'un entier sur 1 octet)

Exemple de sortie :

Petite valeur.

```
Entrez en entier : 42
entier sur 8 octets :          2a (42)
entier sur 4 octets :          2a (42)
entier sur 2 octets :          2a (42)
entier sur 1 octet :          2a (42)
découpe : 00000000_0000_00_2a
```

Exemple de sortie :

Limite sur 4 octets.

```
Entrez en entier : 4000000000
entier sur 8 octets :          ee6b2800 (4000000000)
entier sur 4 octets :          ee6b2800 (4000000000)
entier sur 2 octets :          2800 (10240)
entier sur 1 octet :          0 (0)
découpe : 00000000_ee6b_28_00
```

Exemple de sortie :

Dépassement de capacité du 8 octets.

```
Entrez en entier : 8000000000000000
entier sur 8 octets :      48c273950000 (8000000000000000)
entier sur 4 octets :      73950000 (1939144704)
entier sur 2 octets :      0 (0)
entier sur 1 octet :      0 (0)
découpe : 000048c2_7395_00_00
```

Exercice 5 (..... / 4 points)

Objectif : Décoder un nombre en base quelconque.

- **Entrée 1** : un entier sur 4 octets (base)
- **Entrée 2** : une suite de caractères (nombre)
- **Entrée 3** : un caractère de retour à la ligne (marqueur de fin)

Lire un entier sur 4 octets. Cette entier est la base dans laquelle est représenté le nombre qui suit. Les chiffres de ce nombre dans la base b entière donnée sont représentés de la manière suivante :

- Chiffres de 0 à 9 pour les chiffres de 0 à 9.
- Lettres de a à z (minuscules et majuscules acceptées) pour les chiffres de 10 à 35.
- Format #{valeur} pour tout autre chiffre où valeur est un entier positif inférieur à b .

Pour rappel, un nombre à n chiffres exprimé dans une base b par les chiffres $(c_i)_{i \in [n]}$ comme $(c_1 c_2 c_3 \cdots c_n)_b$ se calcule comme il suit :

$$c_1 b^{n-1} + c_2 b^{n-2} + \cdots + c_{n-1} b^1 + c_n b^0$$

Exemple de sortie :

Base binaire.

```
Entrez une base, puis un nombre dans cette base : 2 101010
Ce nombre vaut 42 en base 10.
```

Exemple de sortie :

Base hexadécimale, grande valeur.

```
Entrez une base, puis un nombre dans cette base : 16 ffffffffffffff
Ce nombre vaut 18446744073709551615 en base 10.
```

Exemple de sortie :

Base 100, avec chiffre non alphanumérique.

```
Entrez une base, puis un nombre dans cette base : 100 4#{42}2
Ce nombre vaut 44202 en base 10.
```

Exemple de sortie :
Base 42, codage de texte court.

```
Entrez une base, puis un nombre dans cette base : 42 exemple
Ce nombre vaut 81204493748 en base 10.
```

B.2.5 S1 2022-2023 : Janvier

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie de faire saisir un entier à l'utilisateur et vérifier s'il correspond à un mot de passe attendu.

Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
Sub Programme

    PassWord = ReadBox "Entrez un mot de passe : "
    Si PassWord = 4242 Alors
        DisplayBox "Bienvenue !"
    Sinon
        DisplayBox "Au revoir !"
        Fin Sub Maintenant
    Fin Si

End Sub
```

Exercice 2 (..... / 4 points)

Objectif : calculatrice d'entiers.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un caractère (opérateur)
- **Entrée 3** : un entier (nombre 2)

Lire un nombre, un opérateur et un autre nombre puis afficher le calcul pour les opérations :

- '+' pour l'addition d'entiers.
- '-' pour la soustraction d'entiers.
- '*' pour la multiplication d'entiers.
- '/' pour la division fractionnaire (à virgule).
- 'd' pour la division entière (quotient de la division euclidienne).
- 'm' pour le modulo (reste de la division euclidienne).

Exemple de sortie :

Addition.

```
>>> 5 + 4  
= 9
```

Exemple de sortie :

Multiplication.

```
>>> 100000000 * 10000000  
= 1000000000000000000
```

Exemple de sortie :

Division fractionnaire.

```
>>> 7 / 5  
= 1.4
```

Exemple de sortie :

Division entière.

```
>>> 7 d 5  
= 1
```

Exemple de sortie :

Modulo.

```
>>> 7 m 5  
= 2
```

Exemple de sortie :

Gestion division par zéro.

```
>>> 7 / 0  
= inf
```

Exercice 3 (..... / 4 points)

Objectif : Simuler un déplacement d'un joueur.

- **Entrée 1** : les coordonnées entières d'un objectif.
- **Entrées suivantes** : des caractères comme commandes au joueur.

Un joueur commence aux coordonnées entières (0, 0). Lire l'objectif à atteindre, puis diriger le joueur par les commandes suivantes :

- 'z' : aller d'une case en haut.
- 's' : aller d'une case en bas.
- 'q' : aller d'une case vers la gauche.
- 'd' : aller d'une case vers la droite.
- '.' : afficher la position du joueur et celle de l'objectif.

Exemple de sortie :

```
Entrez les coordonnées de l'objectif : 2 0
.
joueur : (0, 0)
objectif : (2, 0)
dd
Bravo !
```

Exemple de sortie :

```
Entrez les coordonnées de l'objectif : -5 4
qqs.
joueur : (-2, -1)
objectif : (-5, 4)
qqzzqzz.
joueur : (-5, 3)
objectif : (-5, 4)
z
Bravo !
```

Exercice 4 (..... / 4 points)

Objectif : récupérer la valeur d'un octet.

- **Entrée 1** : un entier sur 8 octets (nombre)
- **Entrée 2** : un entier sur 4 octets (numéro de l'octet)

Lire un entier sur 8 octets puis afficher la valeur de l'octet de cet entier demandé par l'utilisateur.

Exemple de sortie :

```
Entrez en entier : 81985529216486895
Quel octet voulez-vous visualiser ? 1
octet_1(81985529216486895 | 0x0123456789abcdef) = ef
```

Exemple de sortie :

```
Entrez en entier : 81985529216486895
Quel octet voulez-vous visualiser ? 3
octet_3(81985529216486895 | 0x0123456789abcdef) = ab
```

Exemple de sortie :

```
Entrez en entier : 81985529216486895
Quel octet voulez-vous visualiser ? 8
octet_8(81985529216486895 | 0x0123456789abcdef) = 1
```

Exemple de sortie :

```
Entrez en entier : 42
Quel octet voulez-vous visualiser ? 1
octet_1(42 | 0x000000000000002a) = 2a
```

Exercice 5 (..... / 4 points)

Objectif : Gérer une somme de fractions.

- **Entrées :** fraction d'entiers sous la forme `numérateur / dénominateur`

Lire une suite de fractions et en faire la somme. Simplifier si besoin. La suite de fractions se termine par une fraction nulle ou invalide.

Exemple de sortie :

```
Entrez des fractions (zéro pour terminer) : 1 / 2 1 / 3 0 / 0
= 5 / 6 (~ 0.833333)
```

Exemple de sortie :

```
Entrez des fractions (zéro pour terminer) : 4 / 7 10 / 7 0 / 0
= 2
```

Exemple de sortie :

```
Entrez des fractions (zéro pour terminer) : 1 / 2 1 / 3 1 / 5 1 / 7 0 / 0
= 247 / 210 (~ 1.17619)
```

B.2.6 S1 2023-2024 : Alternance

Appréciation

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier individuelle de sa production (✓ calculatrice sans mémoire non programmable autorisée, ✗ documents non autorisés).
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres et sont donnés par ordre croissant de difficulté / notions vues en classe.
- Chaque exercice doit être représenté par un code source complet en **Language C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera, valorisera et pénalisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé en adéquation avec la problématique).
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie de calculer la racine carré d'un entier positif. Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
<?studio.c

let x, y
Read "Entrez un réel : ", x
Si x = 0 Alors
    Afficher "Une solution : 0"
Sinon
    Si x > 0 Alors
        y = racine_carre(x)
        Afficher "Deux solutions : ", -y, " et ", y
    Sinon
        y = racine_carre(-x)
        Afficher "Deux solutions : ", -y, "i et ", y, "i"
    Fin Si
Fin Si

?>
```

Attendus :

- (... / 1) Lancer un programme en Langage C dans un terminal Linux.
- (... / 1) Corps d'un code en Langage C.
- (... / 1) Variables et affichages.
- (... / 1) Expressions et conditions.

Exercice 2 (..... / 4 points)

Objectif : Convertir une adresse IPv4 en un entier.

- **Entrée :** une adresse IPv4 (4 entiers entre 0 et 255)

Une adresse IPv4 est constituée de 4 entiers chacun sur 1 octet. En réalité une telle adresse pourrait se représenter sur un entier de 4 octets. Par exemple, l'adresse IP 127.0.0.1 peut par exemple s'écrire en hexadécimal puis se traduire sous forme d'un entier de 4 octets :

$$(127.0.0.1)_{10} = (7f.00.00.01)_{16}$$

$$(7f)_{16} \times 16^6 + (00)_{16} \times 16^4 + (00)_{16} \times 16^2 + (01)_{16} \times 16^0 = (2130706433)_{10}$$

Entrez une adresse IPv4 : 127.0.0.1
Ceci correspond à l'entier : 2130706433

Entrez une adresse IPv4 : 0.0.1.0
Ceci correspond à l'entier : 256

Entrez une adresse IPv4 : 0.1.0.0
Ceci correspond à l'entier : 65536

Entrez une adresse IPv4 : 1.0.0.0
Ceci correspond à l'entier : 16777216

Entrez une adresse IPv4 : 214.112.45.10
Ceci correspond à l'entier : 3597675786

Attendus :

- (... / 1) Lecture de 4 entiers (chacun d'un octet).
- (... / 1) Lecture formatée (avec les séparateurs).
- (... / 1) Gestion d'entiers non signés.
- (... / 1) Conversion correcte.

Exercice 3 (..... / 4 points)

Objectif : Calculer un PGCD.

- **Entrée 1** : un entier signé sur 8 octets.
- **Entrée 2** : un entier signé sur 8 octets.

Lire deux entiers et énumérer les division euclidiennes successives calculant le plus grand diviseur commun de ces deux entiers :

```
Entrez deux entiers : 12 5
12 = 5 * 2 + 2
5 = 2 * 2 + 1
2 = 1 * 2 + 0
PGCD(12, 5) = 1
```

```
Entrez deux entiers : 5 12
12 = 5 * 2 + 2
5 = 2 * 2 + 1
2 = 1 * 2 + 0
PGCD(5, 12) = 1
```

```
Entrez deux entiers : -1337 -42
1337 = 42 * 31 + 35
42 = 35 * 1 + 7
35 = 7 * 5 + 0
PGCD(-1337, -42) = 7
```

```
Entrez deux entiers : 5 5
5 = 5 * 1 + 0
PGCD(5, 5) = 5
```

```
Entrez deux entiers : 121 1234554321
1234554321 = 121 * 10202928 + 33
121 = 33 * 3 + 22
33 = 22 * 1 + 11
22 = 11 * 2 + 0
PGCD(121, 1234554321) = 11
```

```
Entrez deux entiers : 0 42
PGCD(0, 42) = 42
```

```
Entrez deux entiers : 50000000000000000000 123456789
50000000000000000000 = 123456789 * 4050000036 + 105555596
123456789 = 105555596 * 1 + 17901193
105555596 = 17901193 * 5 + 16049631
17901193 = 16049631 * 1 + 1851562
16049631 = 1851562 * 8 + 1237135
1851562 = 1237135 * 1 + 614427
1237135 = 614427 * 2 + 8281
614427 = 8281 * 74 + 1633
8281 = 1633 * 5 + 116
1633 = 116 * 14 + 9
116 = 9 * 12 + 8
9 = 8 * 1 + 1
8 = 1 * 8 + 0
PGCD(50000000000000000000, 123456789) = 1
```

Attendus :

- (... / 1) Calcul du PGCD.
- (... / 1) Affichage des divisions euclidiennes successives et du résultat.
- (... / 1) Fonctionnement pour saisies diverses d'entiers (négatif, nul).
- (... / 1) Fonctionnement pour des entiers signés sur 8 octets.

Exercice 4 (..... / 4 points)

Objectif : Factorisation d'un entier.

- **Entrée :** un entier signé sur 8 octets.

Un nombre entier peut s'écrire comme le produit de nombres premiers (nombre uniquement divisible par les entiers naturels 1 et lui-même). Lire un entier sur 8 octets et le décomposer en ses facteurs premiers (nous excluons, -1, 0 et 1 de ces facteurs dans la décomposition) :

```
Entrez un enfier : 42
42 = 2 * 3 * 7
```

```
Entrez un enfier : 1620
1620 = 2^2 * 3^4 * 5
```

```
Entrez un enfier : 32767
32767 = 7 * 31 * 151
```

```
Entrez un enfier : 127
127 = 127
```

```
Entrez un enfier : 1
1 = 1
```

```
Entrez un enfier : 0
0 = 0
```

```
Entrez un entier : -1
-1 = -1 * 1
```

```
Entrez un entier : -24
-24 = -1 * 2^3 * 3
```

```
Entrez un entier : 194076479520000000
194076479520000000 = 2^11 * 3^8 * 5^7 * 7^5 * 11
```

```
Entrez un entier : 76333445047
76333445047 = 137 * 5153 * 108127
```

```
Entrez un entier : 643498753646727910
643498753646727910 = 2 * 5 * 17 * 211 * 17939747801693
```

Attendus :

- (... / 1) Calcul exact de la factorisation.
- (... / 1) Affichage identique aux sorties.
- (... / 1) Prise en compte de cas particuliers (négatifs, nul).
- (... / 1) Efficacité algorithmique (temps en $\mathcal{O}(\sqrt{n})$).

Exercice 5 (..... / 4 points)

Objectif : Chiffrer un message.

- **Entrée :** une suite de caractères (terminée par un retour à la ligne)

Oscar a imaginé un chiffrement pour lequel la clé serait basé comme la somme des décalages induits par les lettres précédentes :

- Chiffres de 0 à 9 ajoutent un décalage relatif à 0.
- Lettres de a à z ajoutent un décalage relatif à a.

Texte : A B C 0 7 b ↵
 | ↘ | ↘ | ↘ | ↘ | ↘ | ↘ | ↘ | ↘ |
Clé : 0 0 1 3 3 10 11
 ↓ ↓ ↓ ↓ ↓ ↓ ↓
Résultat : A B D 3 0 1 ↵

```
Saisir du texte : ABC07b
Résultat      : ABD301
```

```
Saisir du texte : 0001000
Résultat      : 0001111
```

```
Saisir du texte : BBBB
Résultat      : BCDEFG
```

```
Saisir du texte : 999999
Résultat      : 987654
```

```
Saisir du texte : 1A1A1A1A
Résultat      : 1B2C3D4E
```

```
Saisir du texte : "010, 010, 010" !
Résultat       : "011, 122, 233" !
```

```
Saisir du texte : Exemple de texte pour ce super Examen !
Résultat       : Ebfrgrv cg dhexb uict mq mgvzq Liiuyl !
```

Attendus :

- (... / 1) Lecture d'une suite de caractères.
- (... / 1) Gestion de l'accumulation (clé).
- (... / 1) Application correcte du procédé.
- (... / 1) Affichage identique aux exemples.

B.2.7 S1 2023-2024 : Janvier initial

Appréciation

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier individuelle de sa production (calculatrice et documents non autorisés).
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres et sont donnés par ordre croissant de difficulté / notions vues en classe.
- Chaque exercice doit être représenté par un code source complet en **Language C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera, valorisera et pénalisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé en adéquation avec la problématique).
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie de comparer deux entiers, mais son programme ne se lance pas en Langage C. Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
import studio

visual studio code :
Lire "Entrez un entier : ", first
Lire "Entrez un entier : ", second
Selon :
    Cas first > second :
        Afficher first, " est plus grand que ", second
    Cas first < second :
        Afficher second " est plus grand que ", first
    Cas first = second :
        Afficher "Les deux entiers sont égaux à ", first
```

Attendus :

- (... / 1) Lancer un programme en Langage C dans un terminal Linux.
- (... / 1) Corps d'un code en Langage C.
- (... / 1) Variables et affichages.
- (... / 1) Expressions et conditions.

Exercice 2 (..... / 4 points)

Objectif : Convertir une adresse IPv4 en un entier.

- **Entrée :** une adresse IPv4 (4 entiers entre 0 et 255)

Une adresse IPv4 est constituée de 4 entiers chacun sur 1 octet. En réalité une telle adresse pourrait se représenter sur un entier de 4 octets. Par exemple, l'adresse IP 127.0.0.1 peut par exemple s'écrire en hexadécimal puis se traduire sous forme d'un entier de 4 octets :

$$(127.0.0.1)_{10} = (7f.00.00.01)_{16}$$

$$(7f)_{16} \times 16^6 + (00)_{16} \times 16^4 + (00)_{16} \times 16^2 + (01)_{16} \times 16^0 = (2130706433)_{10}$$

Entrez une adresse IPv4 : 127.0.0.1
Ceci correspond à l'entier : 2130706433

Entrez une adresse IPv4 : 0.0.1.0
Ceci correspond à l'entier : 256

Entrez une adresse IPv4 : 0.1.0.0
Ceci correspond à l'entier : 65536

Entrez une adresse IPv4 : 1.0.0.0
Ceci correspond à l'entier : 16777216

Entrez une adresse IPv4 : 214.112.45.10
Ceci correspond à l'entier : 3597675786

Attendus :

- (... / 1) Lecture de 4 entiers (chacun d'un octet).
- (... / 1) Lecture formatée (avec les séparateurs).
- (... / 1) Gestion d'entiers non signés.
- (... / 1) Conversion correcte.

Exercice 3 (..... / 4 points)

Objectif : Écrire un entier en binaire.

- Entrée : un entier non signé sur 8 octets.

Lire un entier, puis écrire sa représentation en binaire.

Pour rappel, l'écriture en binaire d'un entier est la suite de chiffres (b_i) dont les valeurs sont soit 0, soit 1. C'est-à-dire qu'un nombre n pourrait s'écrire comme il suit :

$$n = (b_k b_{k-1} \dots b_1 b_0)_2 = b_k 2^k + b_{k-1} 2^{k-1} + \dots + b_1 2^1 + b_0 2^0.$$

Exemple :

$$(42)_{10} = (101010)_2 = 2^5 + 2^3 + 2^1 = 32 + 8 + 2$$

Entrez un entier : 42
 $(42)_{10} = (101010)_2$

```
Entrez un entier : 1  
(1) 10 = (1) 2
```

```
Entrez un entier : 2  
(2) 10 = (10) 2
```

```
Entrez un entier : 3  
(3) 10 = (11) 2
```

```
Entrez un entier : 4
(4)_10 = (100)_2
```

```
Entrez un entier : 8
(8)_10 = (1000)_2
```

```
Entrez un entier : 16
(16)_10 = (10000)_2
```

Attendus :

- (... / 1) Fonctionnement entier non signé sur 8 octets.
- (... / 1) Affichage des chiffres en binaire.
- (... / 1) Affichage des chiffres dans le bon ordre.
- (... / 1) Affichage identique exemples.

Exercice 4 (..... / 4 points)

Objectif : Calculer une moyenne pondérée.

- **Entrées** : couples (réel : note, réel : coefficient) (séparés par une étoile).
- **Fin** : réel négatif.

Un utilisateur souhaite pouvoir saisir une suite de notes associées chacune à un coefficient pour ensuite en calculer la moyenne pondérée. Il terminera sa saisie par une note négative. En supposant une suite de n notes $(x_i)_{i \in [n]}$ et de n coefficients $(c_i)_{i \in [n]}$, leur moyenne pondérée se calculerait par la formule suivante :

$$\bar{x} = \frac{x_1 \times c_1 + x_2 \times c_2 + \dots + x_n \times c_n}{c_1 + c_2 + \dots + c_n}$$

Par exemple :

$$\frac{10 \times 4 + 11 \times 4 + 12 \times 2}{4 + 4 + 2} = 10.8$$

```
Entrez notes * coefficient (jusqu'à une note négative) :  
>>> 10 * 4 11 * 4 12 * 2 -1  
Moyenne : 10.8
```

```
Entrez notes * coefficient (jusqu'à une note négative) :  
>>> 1 * 1 2 * 2 3 * 3 4 * 4 5 * 5 6 * 6 7 * 7 8 * 8 9 * 9 10 * 10  
↪ -1  
Moyenne : 7
```

```
Entrez notes * coefficient (jusqu'à une note négative) :  
>>> -1  
Pas de notes.
```

```
Entrez notes * coefficient (jusqu'à une note négative) :  
>>> 10 * 1  
-1  
Moyenne : 10
```

```
Entrez notes * coefficient (jusqu'à une note négative) :  
>>> 10 * 4  
11 * 1  
-1  
Moyenne : 10.2
```

Attendus :

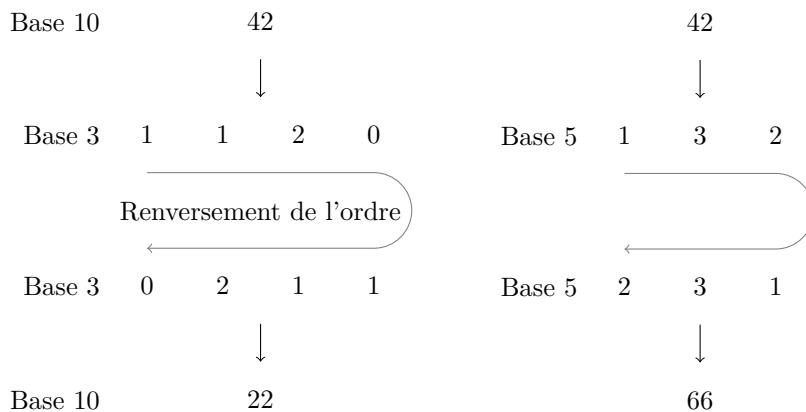
- (... / 1) Lecture de notes et coefficients.
- (... / 1) Arrêt à note négative.
- (... / 1) Prise en compte du cas pas de notes.
- (... / 1) Calcul de moyenne pondérée.

Exercice 5 (..... / 4 points)

Objectif : Renverser un entier dans une base donnée.

- **Entrée 1 :** (nombre à renverser) un entier non signé sur 8 octets.
- **Entrée 2 :** (base de renversement) un entier non signé sur 4 octets.

Oscar cherche à fabriquer une fonction de hachage. Son idée est de renverser les chiffres d'un entier tels qu'ils pourraient être exprimés dans une base donnée :



En effet, $(42)_{10} = (1120)_3$ ($1 \times 27 + 1 \times 9 + 2 \times 3 + 0 \times 1$). En inversant l'ordre des chiffres de $(1120)_3$, nous obtiendrions $(0211)_3 = (22)_{10}$.

```
Entrez un entier : 42 3
Entrez une base pour le renverser : (42)_10 = (1120)_3
(211)_3 = (22)_10
42 renversé en base 3 donne 22
```

```
Entrez un entier : 42 5
Entrez une base pour le renverser : (42)_10 = (132)_5
```

```
(231)_5 = (66)_10
42 renversé en base 5 donne 66
```

```
Entrez un entier : 42 16
Entrez une base pour le renverser : (42)_10 = (2a)_16
(a2)_16 = (162)_10
42 renversé en base 16 donne 162
```

```
Entrez un entier : 42 10
Entrez une base pour le renverser : (42)_10 = (42)_10
(24)_10 = (24)_10
42 renversé en base 10 donne 24
```

```
Entrez un entier : 123456789
Entrez une base pour le renverser : 35
(123456789)_10 = (2c9g1t)_35
(t1g9c2)_35 = (1525332447)_10
123456789 renversé en base 35 donne 1525332447
```

```
Entrez un entier : 10000000000000000000
Entrez une base pour le renverser : 10
(10000000000000000000)_10 = (10000000000000000000)_10
(1)_10 = (1)_10
10000000000000000000 renversé en base 10 donne 1
```

```
Entrez un entier : 10000000000000000000
Entrez une base pour le renverser : 2
(10000000000000000000)_10 =
    ↳ (1101111000010110110101100111010011101100100000000000000000000000)_
(100110111001011100110101101101000001111011)_2 =
    ↳ (2673027240059)_10
10000000000000000000 renversé en base 2 donne 2673027240059
```

```
Entrez un entier : 10000000000000000000
Entrez une base pour le renverser : 3
(10000000000000000000)_10 =
→ (2012221222102101100201020220212020001)_3
(10002021202202010200111012012221222102)_3 =
→ (463001464033139072)_10
10000000000000000000 renversé en base 3 donne 463001464033139072
```

```
Entrez un entier : 10000000000000000000
Entrez une base pour le renverser : 1337
(10000000000000000000)_10 =
→ (#{234}#{91}#{773}#{604}#{137}#{897})_1337
(#{897}#{137}#{604}#{773}#{91}#{234})_1337 =
→ (3832650799361200736)_10
10000000000000000000 renversé en base 1337 donne
→ 3832650799361200736
```

Attendus :

- (... / 1) Affichage d'un entier en base quelconque.
- (... / 1) Renversement dans une base donnée.
- (... / 1) Possibilité de bases diverses.
- (... / 1) Affichage identique aux exemples.

B.2.8 S1 2023-2024 : Janvier alternance

Appréciation

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier individuelle de sa production (calculatrice et documents non autorisés).
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres et sont donnés par ordre croissant de difficulté / notions vues en classe.
- Chaque exercice doit être représenté par un code source complet en **Language C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 5 (boucles) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera, valorisera et pénalisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé en adéquation avec la problématique).
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Adapter un code en langage C.

Un élève débutant en langage C essaie calculer l'exposant d'un réel, mais son programme ne se lance pas en Langage C. Cet élève travaille sur une machine Linux avec les outils vus en cours installés sur sa machine.

Lui indiquer comment lancer son programme depuis un terminal. Puis adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
set Visual Studio C :  
  
a <- prompt("Entrez une valeur : ")  
n <- prompt("Entrez un exposant : ")  
  
display(a ^ n)
```

Attendus :

- (... / 1) Lancer un programme en Langage C dans un terminal Linux.
- (... / 1) Corps d'un code en Langage C.
- (... / 1) Variables et affichages.
- (... / 1) Expressions.

Exercice 2 (..... / 4 points)

Objectif : déterminer un minimum et un maximum.

- **Entrée 1** : un entier (nombre 1)
- **Entrée 2** : un entier (nombre 2)
- **Entrée 3** : un entier (nombre 3)

Lire trois nombres entiers séparés par des virgules puis :

- Afficher le minimum des trois nombres.
- Afficher le maximum des trois nombres.

```
Entrez 3 entiers : 1, 2, 3
min(1, 2, 3) = 1
max(1, 2, 3) = 3
```

```
Entrez 3 entiers : 3, 2, 1
min(3, 2, 1) = 1
max(3, 2, 1) = 3
```

```
Entrez 3 entiers : 5, 7, 2
min(5, 7, 2) = 2
max(5, 7, 2) = 7
```

Attendus :

- (... / 1) Lecture de 3 entiers.
- (... / 1) Lecture formatée séparée par des virgules.
- (... / 1) Calcul du minimum et maximum.
- (... / 1) Affichage du minimum et maximum.

Exercice 3 (..... / 4 points)

Objectif : Diviseurs d'un entier.

- **Entrée :** un entier signé sur 4 octets.

Lire un entier, puis écrire la liste de ses diviseurs séparés par des virgules. Nous avons que a est divisible b si et seulement si le reste de la division euclidienne de a par b est nul.

```
Entrez un entier : 42
Liste des diviseurs de 42 :
1, 2, 3, 6, 7, 14, 21, 42
```

```
Entrez un entier : 127
Liste des diviseurs de 127 :
1, 127
```

```
Entrez un entier : 126
Liste des diviseurs de 126 :
1, 2, 3, 6, 7, 9, 14, 18, 21, 42, 63, 126
```

```
Entrez un entier : 128
Liste des diviseurs de 128 :
1, 2, 4, 8, 16, 32, 64, 128
```

```
Entrez un entier : 1
Liste des diviseurs de 1 :
1
```

Attendus :

- (... / 1) Fonctionnement entier signé sur 4 octets.
- (... / 1) Affichage des diviseurs.
- (... / 1) Affichage des diviseurs dans le bon ordre.
- (... / 1) Affichage identique exemples (avec virgules).

Exercice 4 (..... / 4 points)

Objectif : Algorithme d'Euclide étendu.

- **Entrée 1** : un entier signé sur 4 octets.

- **Entrée 2** : un entier signé sur 4 octets.

Charlie a entendu parler de l'identité de Bézout qui consiste à trouver pour deux entiers a et b dont le plus grand diviseur commun est noté g les valeurs u et v vérifiant l'expression suivante :

$$a \times u + b \times v = g$$

Oscar lui indique qu'il peut poser les suites $(r_i)_{i \in \mathbb{N}}$, $(u_i)_{i \in \mathbb{N}}$, $(v_i)_{i \in \mathbb{N}}$ depuis les données initiales suivantes :

$$r_0 = a, r_1 = b \quad u_0 = 1, u_1 = 0 \quad v_0 = 0, v_1 = 1$$

Puis pour tout entier $i > 1$, calculer r_i , u_i et v_i jusqu'à ce que r_i atteigne 0 par les relations suivantes :

$$r_i = r_{i-2} - \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor r_{i-1} \quad u_i = u_{i-2} - \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor u_{i-1} \quad v_i = v_{i-2} - \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor v_{i-1}$$

Où $\lfloor \cdot \rfloor$ est la partie entière d'un réel. Charlie aimerait pouvoir visualiser ces étapes et donc souhaite que vous réalisiez un programme implémentant ce concept avec le détails à chaque indice i sous la forme suivante de ces suites affichés comme dans les sorties ci-dessous.

$$a \times u_i + b \times v_i = r_i$$

```
Entrez deux entiers : 1337 42
1337 * 1 + 42 * 0 = 1337
1337 * 0 + 42 * 1 = 42
1337 * 1 + 42 * -31 = 35
1337 * -1 + 42 * 32 = 7
```

```
Entrez deux entiers : 610 377
610 * 1 + 377 * 0 = 610
610 * 0 + 377 * 1 = 377
610 * 1 + 377 * -1 = 233
610 * -1 + 377 * 2 = 144
610 * 2 + 377 * -3 = 89
610 * -3 + 377 * 5 = 55
610 * 5 + 377 * -8 = 34
610 * -8 + 377 * 13 = 21
610 * 13 + 377 * -21 = 13
610 * -21 + 377 * 34 = 8
610 * 34 + 377 * -55 = 5
610 * -55 + 377 * 89 = 3
610 * 89 + 377 * -144 = 2
610 * -144 + 377 * 233 = 1
```

```
Entrez deux entiers : 1 0
1 * 1 + 0 * 0 = 1
```

```
Entrez deux entiers : 7 3
7 * 1 + 3 * 0 = 7
7 * 0 + 3 * 1 = 3
7 * 1 + 3 * -2 = 1
```

Attendus :

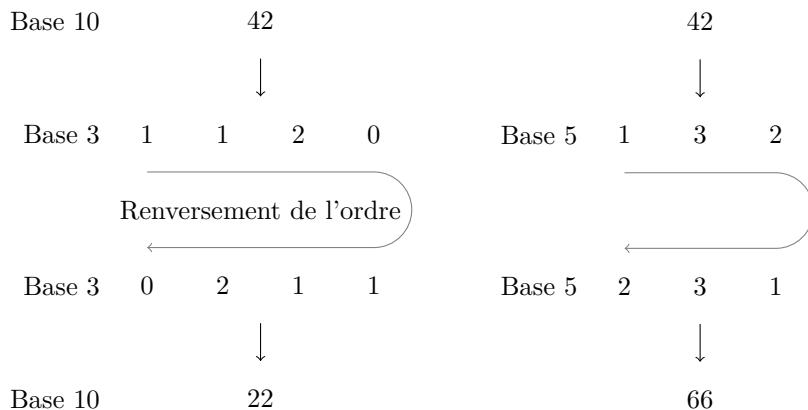
- (... / 1) Calcul du PGCD.
- (... / 1) Modélisation de suites par des variables.
- (... / 1) Calcul de l'identité de Bézout.
- (... / 1) Affichage des étapes de calcul.

Exercice 5 (..... / 4 points)

Objectif : Renverser un entier dans une base donnée.

- **Entrée 1 :** (nombre à renverser) un entier non signé sur 8 octets.
- **Entrée 2 :** (base de renversement) un entier non signé sur 4 octets.

Oscar cherche à fabriquer une fonction de hachage. Son idée est de renverser les chiffres d'un entier tels qu'ils pourraient être exprimés dans une base donnée :



En effet, $(42)_{10} = (1120)_3$ ($1 \times 27 + 1 \times 9 + 2 \times 3 + 0 \times 1$). En inversant l'ordre des chiffres de $(1120)_3$, nous obtiendrions $(0211)_3 = (22)_{10}$.

```
Entrez un entier : 42 3
Entrez une base pour le renverser : (42)_10 = (1120)_3
(211)_3 = (22)_10
42 renversé en base 3 donne 22
```

```
Entrez un entier : 42 5
Entrez une base pour le renverser : (42)_10 = (132)_5
(231)_5 = (66)_10
42 renversé en base 5 donne 66
```

```
Entrez un entier : 42 16
Entrez une base pour le renverser : (42)10 = (2a)16
(2a)16 = (162)10
42 renversé en base 16 donne 162
```

```
Entrez un entier : 42 10
Entrez une base pour le renverser : (42)_10 = (42)_10
(24)_10 = (24)_10
42 renversé en base 10 donne 24
```

```
Entrez un entier : 123456789
Entrez une base pour le renverser : 35
(123456789)_10 = (2c9g1t)_35
(t1g9c2)_35 = (1525332447)_10
123456789 renversé en base 35 donne 1525332447
```

```
Entrez un entier : 10000000000000000000  
Entrez une base pour le renverser : 10  
 $(1000000000000000)_10 = (1000000000000000)_10$   
 $(1)_10 = (1)_10$   
1000000000000000 renversé en base 10 donne 1
```

```
Entrez un entier : 10000000000000000000  
Entrez une base pour le renverser : 3  
 $(10000000000000000000)_10 = (20122212221021011100201020220212020001)_3$   
 $(10002021202202010200111012012221222102)_3 = (463001464033139072)_10$   
10000000000000000000 renversé en base 3 donne 463001464033139072
```

```
Entrez un entier : 10000000000000000000  
Entrez une base pour le renverser : 1337  
 $(10000000000000000000)_10 = (\#234\#\{91\}\#\{773\}\#\{604\}\#\{137\}\#\{897\})_{1337}$   
 $(\#897\#\{137\}\#\{604\}\#\{773\}\#\{91\}\#\{234\})_{1337} = (3832650799361200736)_{10}$   
10000000000000000000 renversé en base 1337 donne 3832650799361200736
```

Attendus :

- (... / 1) Affichage d'un entier en base quelconque.
- (... / 1) Renversement dans une base donnée.
- (... / 1) Possibilité de bases diverses.
- (... / 1) Affichage identique aux exemples.

B.3 Sujets partiels semestre 2

Lors du semestre 2, les étudiants étaient évalués sur les notions des section Fonctions, Tableaux, Pointeur et était considérées comme acquises les sections précédentes.

B.3.1 S2 Sujet zéro

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 3 / 4 / 5 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : déterminer minimum et maximum d'une suite d'entiers.

- **Entrée 1** : séquence d'entiers non nuls
- **Entrée 2** : entier nul (marqueur de fin)

Lire un nombre non déterminé d'entiers non nuls, afficher la valeur du plus petit et du plus grand.

Un entier nul marque la fin de la séquence.

Exemple de sortie :

Entiers positifs.

```
Entrez des valeurs non nulles : 1 2 3 4 5 0
min = 1
max = 5
```

Exemple de sortie :

Petits entiers.

```
Entrez des valeurs non nulles : -1 1 -2 2 0
min = -2
max = 2
```

Exemple de sortie :

Grands entiers.

```
Entrez des valeurs non nulles : 2000000000 -2000000000 0
min = -2000000000
max = 2000000000
```

Exemple de sortie :

Entiers négatifs.

```
Entrez des valeurs non nulles : -100 -10 0
min = -100
max = -10
```

Exercice 2 (..... / 3 points)

Objectif : Calculer une suite numérique.

La suite de Pell est donnée par la relation suivante :

$$\mathcal{P}_n = \begin{cases} 0 & Si\ n = 0 \\ 1 & Si\ n = 1 \\ 2 \times \mathcal{P}_{n-1} + \mathcal{P}_{n-2} & Sinon \end{cases}$$

Votre objectif est d'afficher les 10 premiers éléments de la suite de Pell :

Exemple de sortie :

```
[0, 1, 2, 5, 12, 29, 70, 169, 408, 985]
```

Exercice 3 (..... / 4 points)

Objectif : manipuler des tableaux.

Un collègue a commencé le code commun suivant et vous laisse terminer l'implémentation des fonctionnalités.

```
#include <stdio.h>
#include <stdlib.h>

/* TODO : afficher une liste */
void afficherListe(int liste[]);

/* TODO : concaténer deux listes */
void listeCat(int res[], const int first[], const int second[]);

/* TODO : multiplier chaque élément de la première liste avec chaque
   → élément de la seconde */
void listeProduct(int res[], const int first[], const int second[]);
```

```
int main() {
    int liste1[] = {2, 4, 6, 8, -1};
    int liste2[] = {4, 5, 6, 7, 8, -1};
    int resCat[10];
    int resProd[21];
    printf("liste 1 : ");
    afficherListe(liste1);
    printf("liste 2 : ");
    afficherListe(liste2);
    listeCat(resCat, liste1, liste2);
    printf("liste cat : ");
    afficherListe(resCat);
    listeProduct(resProd, liste1, liste2);
    printf("liste prod : ");
    afficherListe(resProd);
    exit(EXIT_SUCCESS);
}
```

Il s'attend à ce que le résultat soit le suivant :

```
liste 1 : [2, 4, 6, 8]
liste 2 : [4, 5, 6, 7, 8]
liste cat : [2, 4, 6, 8, 4, 5, 6, 7, 8]
liste prod : [8, 10, 12, 14, 16, 16, 20, 24, 28, 32, 24, 30, 36, 42, 48,
              → 32, 40, 48, 56, 64]
```

Exercice 4 (..... / 5 points)

Objectif : concaténer des chaînes de caractères.

Un élève débutant en langage C a vu qu'il peut être simple de concaténer deux chaînes de caractères avec d'autres langages de programmation.

Cependant le compilateur C refuse de le laisser additionner deux chaînes et il ne comprend pas l'erreur :

```
exo4_err.c: In function 'spaceCat':  
exo4_err.c:8:15: error: invalid operands to binary + (have 'const  
→ char *' and 'const char *')  
    return first + second;  
           ^
```

Expliquer la problématique.

Puis, adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
#include <stdio.h>  
#include <stdlib.h>  
  
char * spaceCat(  
    const char * first,  
    const char * second) {  
    first += ' ';  
    return first + second;  
}  
  
int main() {  
    printf("%s !\n", spaceCat("Hello", "ESGI"));  
    exit(EXIT_SUCCESS);  
}
```

Exercice 5 (..... / 4 points)

Objectif : afficher un nombre en bases de 2 à 16.

- **Entrée :** un entier (nombre)

Lire un nombre, puis l'afficher dans les base allant de la base 2 à la base 16.

Exemple de sortie :

1.

```
Entrez une valeur : 1
(1)_2
(1)_3
(1)_4
(1)_5
(1)_6
(1)_7
(1)_8
(1)_9
(1)_10
(1)_11
(1)_12
(1)_13
(1)_14
(1)_15
(1)_16
```

Exemple de sortie :

16.

```
Entrez une valeur : 16
(10000)_2
(121)_3
(100)_4
(31)_5
(24)_6
(22)_7
(20)_8
(17)_9
(16)_10
(15)_11
(14)_12
(13)_13
(12)_14
(11)_15
(10)_16
```

Exemple de sortie :

42.

```
Entrez une valeur : 42
(101010)_2
(1120)_3
(222)_4
(132)_5
(110)_6
(60)_7
(52)_8
(46)_9
(42)_10
(39)_11
(36)_12
(33)_13
(30)_14
(2c)_15
(2a)_16
```

Exemple de sortie :

800000000.

```
Entrez une valeur : 8000000000
(11011100110101100101000000000000)_2
(20212211210200220022)_3
(13130311211000000)_4
(112341000000000)_5
(3401455433012)_6
(402150610106)_7
(73465450000)_8
(22575362808)_9
(8000000000)_10
(3435878453)_11
(1673225768)_12
(9a6543115)_13
(55c6a7c76)_14
(31c4ea585)_15
(1dc65000)_16
```

B.3.2 S2 2021-2022 : Alternance

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 3 / 3 / 4 / 6

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Connaître le langage C et identifier des erreurs.

Pour démontrer le concept des boucles en langage C, un camarade propose les codes suivants. Chaque code doit afficher les entiers de 0 à 4.

- Validez ou non les codes proposés.
- Dans le cas où vous l'invalidez, indiquez le **problème rencontré**.
- Puis proposez une version **corrigée** du programme en C ANSI.

```
/* 1) Boucle Pour */
#include <stdio.h>
#include <stdlib.h>

int main() {
    for(i = 0; i < 5; ++i) {
        printf("%d\n", i);
    }
    exit(EXIT_SUCCESS);
}
```

```
/* 2) Boucle Tant que */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;
    while(i < 5) {
        printf("%d\n", i);
    }
    exit(EXIT_SUCCESS);
}
```

```
/* 3) Boucle Faire Tant que */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;
    loop {
        printf("%d\n", i);
    } while i < 5
    exit(EXIT_SUCCESS);
}
```

Exercice 2 (..... / 3 points)

Objectif : Déterminer une moyenne avec coefficients.

- **Entrée 1** : séquence de couples (note, coefficient) dont note et coefficient sont séparés par une étoile
- **Entrée 2** : valeur négative (marqueur de fin)

Lire une suite de taille non déterminée de notes suivies d'une étoile et de leur coefficient.
Puis en calculer la moyenne.

Une valeur négative marque la fin de la séquence.

Rappel : une moyenne de valeurs $(x_i)_{i \in [n]}$ avec coefficients $(c_i)_{i \in [n]}$ se calcule de la manière suivante :

$$\bar{x} = \frac{x_1 \times c_1 + x_2 \times c_2 + \dots + x_n \times c_n}{c_1 + c_2 + \dots + c_n}$$

Exemple de sortie :

Coefficients différents :

```
Entrez une suite de (note * coeff) :
10 * 2 20 * 1 5 * 4 12 * 2 -1
Moyenne : 9.33333
```

$$\frac{10 \times 2 + 20 \times 1 + 5 \times 4 + 12 \times 2}{2 + 1 + 4 + 2} \simeq 9.33333$$

Exemple de sortie :

Mêmes coefficients :

```
Entrez une suite de (note * coeff) :
10 * 1 11 * 1 12 * 1 -1
Moyenne : 11
```

$$\frac{10 \times 1 + 11 \times 1 + 12 \times 1}{1 + 1 + 1} = 11$$

Exercice 3 (..... / 3 points)

Objectif : Calculer une suite numérique.

La suite de Fibonacci est donnée par la relation suivante :

$$\mathcal{F}_n = \begin{cases} 0 & Si\ n = 0 \\ 1 & Si\ n = 1 \\ \mathcal{F}_{n-1} + \mathcal{F}_{n-2} & Sinon \end{cases}$$

Votre objectif est d'afficher les 10 premiers éléments de la suite de Fibonacci.
Puis si vous supposez que votre méthode ne donnerait pas les 100 premiers éléments de cette suite dans un temps raisonnable, proposez une méthode qui le permettrait.

Exemple de sortie :

```
[0, 1, 2, 5, 12, 29, 70, 169, 408, 985]
```

Exercice 4 (..... / 4 points)

Objectif : Manipuler des chaînes de caractères.

Un collègue a commencé le code commun suivant et vous laisse terminer l'implémentation des fonctionnalités.

```
#include <stdio.h>
#include <stdlib.h>

/* passe chaque lettre en majuscule */
void makeUpper(char * chaine);

/* passe chaque lettre en minuscule */
void makeLower(char * chaine);

/* affiche le nombre d'occurrence de chaque lettre */
```

```
void afficheOccurrencesLettres(char * chaine);

int main() {
    char texte[] = "Bienvenue a l'examen de Langage C";
    makeUpper(texte);
    printf("%s\n", texte);
    makeLower(texte);
    printf("%s\n", texte);
    afficheOccurrencesLettres(texte);
    exit(EXIT_SUCCESS);
}
```

Il s'attend à ce que le résultat soit le suivant :

```
BIENVENUE A L'EXAMEN DE LANGAGE C
bienvenue a l'examen de langage c
'a' : 4
'b' : 1
'c' : 1
'd' : 1
'e' : 7
'g' : 2
'i' : 1
'l' : 2
'm' : 1
'n' : 4
'u' : 1
've' : 1
'x' : 1
```

Exercice 5 (..... / 6 points)

Objectif : Gérer une liste dynamique, la trier et y rechercher.

Un ami aimerait tenir un historique d'une suite de numéros et vérifier si un nouveau numéro est présent dans cette liste. Il a commencé un programme mais a rapidement abandonné. Complétez le code suivant :

```
#include <stdio.h>
#include <stdlib.h>

int * lireListe(int * taille);

void afficherListe(const int * liste, int taille);

void trierListe(int * liste, int taille);

void swap(int * a, int * b);

int recherche(int valeur, const int * liste, int taille);

int main() {
    int * liste = lireListe(/* ??? */);
    /* Les pointeurs c'est trop difficile, need help ! */
    exit(EXIT_SUCCESS);
}
```

Il s'attend à ce que le résultat soit le suivant :

```
12335 54875 65264 13347 98566 94585 48514 -1
[12335, 54875, 65264, 13347, 98566, 94585, 48514]
[12335, 13347, 48514, 54875, 65264, 94585, 98566]
Quel numéro rechercher ?
>>> 13347
Numéro trouvé.
```

B.3.3 S2 2021-2022 : Janvier

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 3 / 4 / 4 / 5

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Connaître le langage C et identifier des erreurs.

Pour démontrer le concept des boucles en langage C, un camarade propose les codes suivants. Chaque code doit afficher les entiers de 0 à 4.

- Validez ou non les codes proposés.
- Dans le cas où vous l'invalidez, indiquez le **problème rencontré**.
- Puis proposez une version **corrigée** du programme en C ANSI.

```
/* 1) Boucle Pour */
#include <stdio.h>
#include <stdlib.h>

int main() {
    for(i = 0; i < 5; ++i) {
        printf("%d\n", i);
    }
    exit(EXIT_SUCCESS);
}
```

```
/* 2) Boucle Tant que */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;
    while(i < 5) {
        printf("%d\n", i);
    }
    exit(EXIT_SUCCESS);
}
```

```
/* 3) Boucle Faire Tant que */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i;
    loop {
        printf("%d\n", i);
    } while i < 5
    exit(EXIT_SUCCESS);
}
```

Exercice 2 (..... / 3 points)

Objectif : Calculer une suite numérique.

La suite de Fibonacci est donnée par la relation suivante :

$$\mathcal{F}_n = \begin{cases} 0 & Si\ n = 0 \\ 1 & Si\ n = 1 \\ \mathcal{F}_{n-1} + \mathcal{F}_{n-2} & Sinon \end{cases}$$

Votre objectif est d'afficher les 10 premiers éléments de la suite de Fibonacci.
Puis si vous supposez que votre méthode ne donnerait pas les 100 premiers éléments de cette suite dans un temps raisonnable, proposez une méthode qui le permettrait.

Exemple de sortie :

```
[0, 1, 2, 5, 12, 29, 70, 169, 408, 985]
```

Exercice 3 (..... / 4 points)

Objectif : Manipuler une chaîne de caractères.

- **Entrée :** Un mot

Lire un mot puis lui appliquer les traitements suivants :

- Mettre en majuscules.
- Changer le sens de lecture : renverser la chaîne de caractères.
- Mélanger les lettres aléatoirement.

Exemple de sortie :

Nom : Fibonacci.

```
Entrez un nom : Fibonacci
majuscules : FIBONACCI
renversé : ICCANOBIF
mélangé : BAOICNICF
```

Exercice 4 (..... / 4 points)

Objectif : Calculer une exponentiation matricielle 2x2.

Un matheux vous affirme que la suite de Fibonacci peut se calculer comme la mise à une puissance donnée de la matrice $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ et par conséquent que ceci demanderait un nombre d'étapes logarithmique pour ce calcul.

A priori, une matrice 2x2 serait un tableau à deux dimensions. Il vous précise aussi comment calculer la multiplication entre deux matrices :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

Proposer un programme en langage C qui calcule les puissances de $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ jusqu'à la puissance 9 :

```
|   1   0 |
|   0   1 |
---
|   1   1 |
|   1   0 |
---
|   2   1 |
|   1   1 |
---
|   3   2 |
|   2   1 |
---
...
---
|  21  13 |
| 13   8 |
---
|  34  21 |
| 21  13 |
---
|  55  34 |
| 34  21 |
```

Exercice 5 (..... / 5 points)

Objectif : Mini-interpréteur sur liste d'entiers.

- **Entrée 1** : Un entier (taille de la plage mémoire)
- **Entrée 2** : Nombre non déterminé de caractères (commandes)

Un ami a entendu parler du langage Brainfuck. C'est un langage où vous pouvez vous déplacer dans la mémoire et y changer des valeurs. Il aimerait que vous en codiez un mini-interpréteur dans une plage de taille donnée par l'utilisateur en suivant les règles suivantes :

- '+' ajoute 1 à la case mémoire regardée par le curseur.
- '-' retire 1 à la case mémoire regardée par le curseur.
- '=' égalise toutes les cases mémoire à la valeur de la case mémoire regardée.
- '>' déplace le curseur vers la droite (retour au début si dépasse de la plage mémoire).
- '<' déplace le curseur vers la gauche (retour à la fin si dépassé de la plage mémoire).
- '.' affiche le contenu de la plage mémoire.

```
taille de la mémoire : 4
.
[0, 0, 0, 0]
+.
[1, 0, 0, 0]
=.
[1, 1, 1, 1]
>+.
[1, 3, 1, 1]
=.
[3, 3, 3, 3]
<--.
[1, 3, 3, 3]
<++++.
[1, 3, 3, 7]
-->>-
[1, 2, 3, 5]
```

B.3.4 S2 2022-2023

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier de sa production.
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres.
- Chaque exercice doit être représenté par un code source en **langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera et valorisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé).
- Les exercices sont ordonnés par ordre croissant en fonction des notions vues en classe et le temps / complexité qui peut être requis pour atteindre les objectifs de chaque exercice.
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Le sujet est à rendre avec la copie pour notation.

Bon courage !

Exercice 1 (..... / 4 points)

Objectif : Calculer une suite numérique.

La factorielle est donnée par la relation suivante :

$$n! = \begin{cases} 1 & Si\ n = 0 \\ n \times (n - 1)! & Sinon \end{cases}$$

Votre objectif est d'afficher les 16 premiers éléments de la factorielle.

Vous devrez coder une fonction `factorielle` et l'utiliser dans votre programme.

Notez, par exemple, que le calcul de 5! est le suivant :

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Exemple de sortie :

```
0 ! = 1
1 ! = 1
2 ! = 2
3 ! = 6
4 ! = 24
5 ! = 120
6 ! = 720
7 ! = 5040
8 ! = 40320
9 ! = 362880
10 ! = 3628800
11 ! = 39916800
12 ! = 479001600
13 ! = 6227020800
14 ! = 87178291200
15 ! = 1307674368000
```

Exercice 2 (..... / 4 points)

Objectif : Déterminer une moyenne avec coefficients.

- **Entrée 1** : séquence de couples (note, coefficient) dont note et coefficient sont séparés par une étoile
- **Entrée 2** : valeur négative (marqueur de fin)

Lire une suite de taille non déterminée de notes suivies d'une étoile et de leur coefficient.
Puis en calculer la moyenne.

Une valeur négative marque la fin de la séquence.

Rappel : une moyenne de valeurs $(x_i)_{i \in [n]}$ avec coefficients $(c_i)_{i \in [n]}$ se calcule de la manière suivante :

$$\bar{x} = \frac{x_1 \times c_1 + x_2 \times c_2 + \dots + x_n \times c_n}{c_1 + c_2 + \dots + c_n}$$

Exemple de sortie :

Coefficients différents :

```
Entrez une suite de (note * coeff) :
10 * 2 20 * 1 5 * 4 12 * 2 -1
Moyenne : 9.33333
```

$$\frac{10 \times 2 + 20 \times 1 + 5 \times 4 + 12 \times 2}{2 + 1 + 4 + 2} \simeq 9.33333$$

Exemple de sortie :

Mêmes coefficients :

```
Entrez une suite de (note * coeff) :
10 * 1 11 * 1 12 * 1 -1
Moyenne : 11
```

$$\frac{10 \times 1 + 11 \times 1 + 12 \times 1}{1 + 1 + 1} = 11$$

Exercice 3 (..... / 4 points)

Objectif : concaténer des chaînes de caractères.

Un élève débutant en langage C a vu qu'il peut être simple de concaténer des chaînes de caractères avec d'autres langages de programmation.

Cependant le compilateur C refuse de le laisser additionner ces chaînes et il ne comprend pas l'erreur :

```
exo3_err.c: In function 'main':  
exo3_err.c:9:34: error: invalid operands to binary + (have 'char  
→ *' and 'char *')  
    char * full_name = prenom + ' ' + nom;  
                           ~~~~~ ^
```

Expliquer la problématique.

Puis, adapter le code suivant en langage C pour qu'il soit fonctionnel :

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
    char nom[100], prenom[50];  
    printf("Bonjour, entrez votre nom et prénom : ");  
    scanf("%s %s", nom, prenom);  
  
    char * full_name = prenom + ' ' + nom;  
  
    printf("Vous êtes donc %s !\n", full_name);  
  
    exit(EXIT_SUCCESS);  
}
```

Exercice 4 (..... / 4 points)

Objectif : Calculer une exponentiation matricielle 2x2.

Un matheux vous affirme que la suite de Fibonacci peut se calculer comme la mise à une puissance donnée de la matrice $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ et par conséquent que ceci demanderait un nombre d'étapes logarithmique pour ce calcul.

A priori, une matrice 2x2 serait un tableau à deux dimensions. Il vous précise aussi comment calculer la multiplication entre deux matrices :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

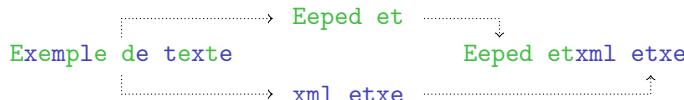
Proposer un programme en langage C qui calcule les puissances de $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ jusqu'à la puissance 9 :

```
| 1 0 |
| 0 1 |
---
| 1 1 |
| 1 0 |
---
| 2 1 |
| 1 1 |
---
| 3 2 |
| 2 1 |
---
...
---
| 21 13 |
| 13 8 |
---
| 34 21 |
| 21 13 |
---
| 55 34 |
| 34 21 |
```

Exercice 5 (..... / 4 points)

Objectif : Lecture en deux à n temps.

Un ami a trouvé un moyen de rendre non triviale la lecture d'un texte tout en étant capable de le lire quand même à vue. Ceci consisterait à lire une lettre sur deux et en faire deux suites de lettres :



Proposez un programme qui code un texte lu au clavier selon cette méthode et vérifie que le chiffrement est correcte. Puis généralisez la méthode pour qu'elle fonctionne pour une lecture toutes les **n** (entier) valeurs.

Exemple de sortie :

Exemple donné (lecture toutes les deux lettres) :

```
Entrez une clé : 2
Entrez une ligne de texte :
Exemple de texte
>>> input :
"Exemple de texte"
>>> code :
"Eeped etxml etxe"
>>> decode :
"Exemple de texte"
```

Exemple de sortie :

Lecture toutes les trois lettres :

```
Entrez une clé : 3
Entrez une ligne de texte :
Exemple de texte
>>> input :
"Exemple de texte"
>>> code :
"Emeeeexp xeldtt"
>>> decode :
"Exemple de texte"
```

B.3.5 S2 2023-2024 : Alternance

Appréciation

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier individuelle de sa production (calculatrice et documents non autorisés).
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres et sont donnés par ordre croissant de difficulté / notions vues en classe.
- Chaque exercice doit être représenté par un code source complet en **Langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera, valorisera et pénalisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé en adéquation avec la problématique).
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Cette page et la grille de notation sont à rendre avec la copie.

Bon courage !

Grille de notation

Exercice 1 (..... / 4 points)

- (... / 1) Boucles.
- (... / 1) Fonctions.
- (... / 1) Chaînes de caractères.
- (... / 1) Pointeurs.

Exercice 2 (..... / 4 points)

- (... / 1) Écriture d'une fonction.
- (... / 1) Appel d'une fonction.
- (... / 1) Fonction récursive.
- (... / 1) Calcul de puissance rapide.

Exercice 3 (..... / 4 points)

- (... / 1) Lecture de caractères en un parcours.
- (... / 1) Affichage lexicographique des occurrences.
- (... / 1) Fonctionnement : comptage d'occurrences de lettres.
- (... / 1) Efficace en temps et en mémoire.

Exercice 4 (..... / 4 points)

- (... / 1) Lecture de deux listes d'entiers.
- (... / 1) Trier une liste d'entiers ou effectuer un traitement pertinent / efficace en temps et mémoire.
- (... / 1) Afficher une liste d'entiers.
- (... / 1) Vérifier si une liste est anagramme de l'autre.

Exercice 5 (..... / 4 points)

- (... / 2) Fonctionnement récursif.
- (... / 2) Fonctionnement optimisé.

Exercice 1 (..... / 4 points)

Objectif : Valider un fonctionnement.

Indiquer si les codes suivants comportent des anomalies. Puis :

- si le code est valide, indiquer ce qu'il fait.
- si le code est invalide, proposer une correction.

```
/* 1) */
#include <stdio.h>

int main() {
    int i;
    while(i < 10)
        printf("%d\n", i);
    return 0;
}
```

```
/* 2) */
#include <stdio.h>
#include <stdlib.h>

int main() {
    char array[] = "ESGI";
    printf("%c\n", array);
    exit(EXIT_SUCCESS);
}
```

```
/* 3) */
#include <stdio.h>

int carre(int x) {
    carre = x * x;
}

int main() {
    printf("%d\n", carre(42));
    return 0;
}
```

```
/* 4) */
#include <stdio.h>
#include <stdlib.h>

int main() {
    int entier = 2;
    char * ptr = (char *)&entier;
    (entier - 1)[ptr] = *ptr;
    printf("%d\n", entier);
    exit(EXIT_SUCCESS);
}
```

Exercice 2 (..... / 4 points)

Objectif : Puissance rapide récursive.

- **Entrée 1** : base (entier sur 8 octets)
- **Entrée 2** : exposant (entier sur 8 octets)

Le calcul d'une puissance d'une base entière α pourrait se calculer efficacement depuis une expression récursive : dans l'idée, en calculant la puissance de la moitié de son exposant entier n , puis mettant le résultat au carré :

$$\alpha^n = \begin{cases} 0 & \text{Si } n < 0 \\ 1 & \text{Si } n = 0 \\ \left(\alpha^{\lfloor \frac{n}{2} \rfloor}\right)^2 & \text{Si } n \text{ est pair} \\ \alpha \left(\alpha^{\lfloor \frac{n}{2} \rfloor}\right)^2 & \text{Si } n \text{ est impair} \end{cases}$$

```
Entrez un entier : 5
Entrez un exposant : -1
5^-1 = 0
```

```
Entrez un entier : 5
Entrez un exposant : 0
5^0 = 1
```

```
Entrez un entier : 5
Entrez un exposant : 1
5^1 = 5
```

```
Entrez un entier : 5
Entrez un exposant : 2
5^2 = 25
```

```
Entrez un entier : 5
Entrez un exposant : 3
5^3 = 125
```

```
Entrez un entier : 2
Entrez un exposant : 42
2^42 = 4398046511104
```

```
Entrez un entier : 2
Entrez un exposant : 60
2^60 = 1152921504606846976
```

```
Entrez un entier : 2
Entrez un exposant : 61
2^61 = 2305843009213693952
```

Exemple de fonctionnement :

$$\begin{array}{l|l} \begin{aligned} 2^{42} &= \left(2^{\lfloor \frac{42}{2} \rfloor}\right)^2 = (2^{21})^2 \\ 2^{21} &= 2 \left(2^{\lfloor \frac{21}{2} \rfloor}\right) = 2 (2^{10})^2 \\ 2^{10} &= \left(2^{\lfloor \frac{10}{2} \rfloor}\right)^2 = (2^5)^2 \\ 2^5 &= 2 \left(2^{\lfloor \frac{5}{2} \rfloor}\right)^2 = 2 (2^2)^2 \\ 2^2 &= \left(2^{\lfloor \frac{2}{2} \rfloor}\right)^2 = (2^1)^2 \end{aligned} & \begin{aligned} 2^1 &= 2 \left(2^{\lfloor \frac{1}{2} \rfloor}\right)^2 = 2 (2^0)^2 \\ 2^0 &= 1 \\ 2^1 &= 2 \times 1 = 2 \\ 2^2 &= 2^2 = 4 \\ 2^5 &= 2 \times 4^2 = 32 \\ 2^{10} &= 32^2 = 1\,024 \\ 2^{21} &= 2 \times 1\,024^2 = 2\,097\,152 \\ 2^{42} &= 2\,097\,152^2 = 4\,398\,046\,511\,104 \end{aligned} \end{array}$$

Exercice 3 (..... / 4 points)

Objectif : Compter des occurrences de lettres.

- **Entrée :** une ligne de caractères (suite de caractères marquée comme terminée par un retour à la ligne).

Compter le nombre d'occurrences de chaque lettre alphabétique et le nombre de lettres non alphabétiques pour la saisie par l'utilisateur d'une ligne de caractères.

```
Entrez du texte : Exemple
'e' : 3 fois
'l' : 1 fois
'm' : 1 fois
'p' : 1 fois
'x' : 1 fois
```

```
Entrez du texte : Texte
'e' : 2 fois
't' : 2 fois
'x' : 1 fois
```

```
Entrez du texte : Hello ESGI !!!!
'e' : 2 fois
'g' : 1 fois
'h' : 1 fois
'i' : 1 fois
'l' : 2 fois
'o' : 1 fois
's' : 1 fois
autres : 6 fois
```

Exercice 4 (..... / 4 points)

Objectif : Vérification d'anagrammes.

- **Entrée 1** : une suite d'entiers positifs (maximum : 100 entiers, terminée par un entier négatif).
- **Entrée 2** : une suite d'entiers positifs (maximum : 100 entiers, terminée par un entier négatif).

Charlie souhaite vérifier si deux listes sont des anagrammes. Deux listes sont des anagrammes si il existe une permutation de l'une vers l'autre :

- [1, 2, 3] et [3, 1, 2] sont des anagrammes car ils ont pour chaque entier le même nombre d'occurrences (en changeant l'ordre des nombres de [1, 2, 3], nous pouvons obtenir la liste [3, 1, 2]).
- [1, 1, 2, 3] et [2, 2, 1, 3] ne sont pas des anagrammes car 1 apparaît par exemple deux fois dans la première liste et une fois dans la seconde.
- [1, 5, 2] et [3, 1, 2] ne sont pas des anagrammes car 5 n'est pas présent dans la seconde liste (et 3 n'est pas présent dans la première).

Proposer un code qui lit deux listes d'entiers positifs puis indique si ce sont des anagrammes. Notez que la liste peut être composée de grandes valeurs : Charlie vous indique que le problème est trivial pour deux listes triées. Votre code doit produire un résultat identique aux sorties suivantes :

```
Entrez une liste de valeurs : 1 2 3 -1
Entrez une liste de valeurs : 3 1 2 -1
[1, 2, 3] et [3, 1, 2] sont des anagrammes.
```

```
Entrez une liste de valeurs : 1 1 2 3 -1
Entrez une liste de valeurs : 2 2 1 3 -1
[1, 1, 2, 3] et [2, 2, 1, 3] ne sont pas des anagrammes.
```

```
Entrez une liste de valeurs : 1 5 2 -1
Entrez une liste de valeurs : 3 1 2 -1
[1, 5, 2] et [3, 1, 2] ne sont pas des anagrammes.
```

```
Entrez une liste de valeurs : 1 3 7 -1
Entrez une liste de valeurs : 1 3 3 7 -1
[1, 3, 7] et [1, 3, 3, 7] ne sont pas des anagrammes.
```

```
Entrez une liste de valeurs : 2000000000 1000000000 0 -1
Entrez une liste de valeurs : 1000000000 0 2000000000 -1
[2000000000, 1000000000, 0] et [1000000000, 0, 2000000000] sont
→ des anagrammes.
```

```
Entrez une liste de valeurs : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
→ 16 17 18 19 20 -1
Entrez une liste de valeurs : 2 4 6 8 10 12 14 16 18 20 19 17 15
→ 13 11 9 7 5 3 1 -1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
→ 19, 20] et [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 19, 17, 15,
→ 13, 11, 9, 7, 5, 3, 1] sont des anagrammes.
```

Exercice 5 (..... / 4 points)

Objectif : Suite de Fibonacci généralisée.

- **Entrée :** le degré de la suite (un entier)

Oscar s'intéresse à la généralisation de la suite de Fibonacci. Il a découvert la suite de Tribonacci, la suite de Tétranacci et qu'elles semblent se généraliser. Il vous indique que les suites énoncées peuvent s'écrire pour tout entier positif n par les relations suivantes :

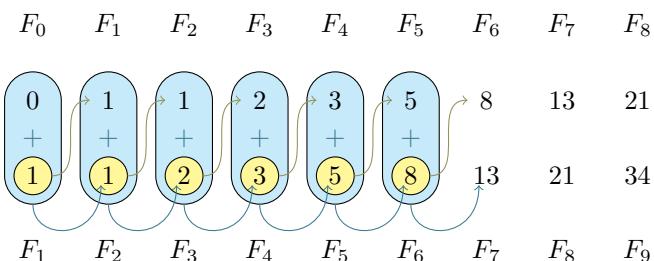
- Fibonacci (degré 2) :

$$F_n = \begin{cases} 0 & \text{Si } n = 0 \\ 1 & \text{Si } n = 1 \\ F_{n-1} + F_{n-2} & \text{Sinon} \end{cases}$$

Pour exemples :

$$\left| \begin{array}{lcl} F_2 & = & F_{2-1} + F_{2-2} \\ & = & F_1 + F_0 \\ & = & 1 + 0 \\ & = & 1 \end{array} \right| \left| \begin{array}{lcl} F_3 & = & F_{3-1} + F_{3-2} \\ & = & F_2 + F_1 \\ & = & 1 + 1 \\ & = & 2 \end{array} \right| \left| \begin{array}{lcl} F_5 & = & F_{5-1} + F_{5-2} \\ & = & F_4 + F_3 \\ & = & 3 + 2 \\ & = & 5 \end{array} \right|$$

Conceptuellement ceci se calcule depuis les deux termes précédents :



(Tourner la page pour la suite)

- Tribonacci (degré 3 : addition des 3 termes précédents) :

$$A_n = \begin{cases} 0 & \text{Si } n < 2 \\ 1 & \text{Si } n = 2 \\ A_{n-1} + A_{n-2} + A_{n-3} & \text{Sinon} \end{cases}$$

- Tétranacci (degré 4 : addition des 4 termes précédents) :

$$B_n = \begin{cases} 0 & \text{Si } n < 3 \\ 1 & \text{Si } n = 3 \\ B_{n-1} + B_{n-2} + B_{n-3} + B_{n-4} & \text{Sinon} \end{cases}$$

- Généralisation (degré d : addition des d termes précédents) :

$$F_{d,n} = \begin{cases} 0 & \text{Si } n < d - 1 \\ 1 & \text{Si } n = d - 1 \\ \underbrace{F_{d,n-1} + F_{d,n-2} + \cdots + F_{d,n-d}}_{d \text{ éléments}} & \text{Sinon} \end{cases}$$

Afficher les 20 premières valeurs de chaque suite en fonction du degré demandé.
Votre méthode sera considérée optimisée si elle permet de calculer le terme d'indice 100 en temps et en mémoire raisonnables.

```
Degré de la suite : 1
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
Degré de la suite : 2
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
 ↵ 1597, 2584, 4181]
```

```
Degré de la suite : 3
[0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705,
 ↵ 3136, 5768, 10609, 19513]
```

```
Degré de la suite : 4
[0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, 208, 401, 773, 1490,
↪ 2872, 5536, 10671, 20569]
```

```
Degré de la suite : 5
[0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, 120, 236, 464, 912, 1793,
↪ 3525, 6930, 13624]
```

```
Degré de la suite : 8
[0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 64, 128, 255, 509,
↪ 1016, 2028]
```

B.3.6 S2 2023-2024 : Janvier initial

Appréciation

Note

..... / 20

Prénom NOM :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Exercice 5
..... / 4 / 4 / 4 / 4 / 4

Modalités

- L'étudiant a 1 heure et 30 minutes pour composer sur ce sujet et rendre une copie papier individuelle de sa production (calculatrice et documents non autorisés).
- Il est conseillé de lire l'intégralité du sujet avant de composer : les 5 exercices peuvent être traités indépendamment les uns des autres et sont donnés par ordre croissant de difficulté / notions vues en classe.
- Chaque exercice doit être représenté par un code source complet en **Langage C** tel qu'il pourrait être fourni à un compilateur.
- Les attendus à utiliser pour réussir les exercices ne dépassent pas les notions étudiées dans le support de cours jusqu'à la section 8 (pointeurs) : l'utilisation de notions externes au cours ne donne pas de points supplémentaires et n'est pas nécessaire (au risque de se complexifier une tâche qui peut être simple).
- L'évaluation vérifiera, valorisera et pénalisera des éléments de code (entrées, sorties, logique conditionnelle, validité d'opérations, inclusions et autres éléments pertinents liés au langage C et à la logique du code proposé en adéquation avec la problématique).
- Des exemples de sorties peuvent être donnés pour aider à comprendre la logique attendue par l'exercice.
- Cette page et la grille de notation sont à rendre avec la copie.

Bon courage !

Grille de notation

Exercice 1 (..... / 4 points)

- (... / 1) Boucles.
- (... / 1) Fonctions.
- (... / 1) Chaînes de caractères.
- (... / 1) Pointeurs.

Exercice 2 (..... / 4 points)

- (... / 1) Écriture d'une fonction.
- (... / 1) Appel d'une fonction.
- (... / 1) Fonction récursive.
- (... / 1) Affichage triangle de Pascal.

Exercice 3 (..... / 4 points)

- (... / 1) Lecture de caractères en un parcours.
- (... / 1) Sauvegarde d'éléments recherchés.
- (... / 1) Affichage des adversaires.
- (... / 1) Affichage des distances au joueurs.

Exercice 4 (..... / 4 points)

- (... / 1) Représentation de chaînes de caractères.
- (... / 1) Lecture de deux chaînes de caractères.
- (... / 1) Manipuler une chaîne de caractères.
- (... / 1) Vérifier si un mot est anagramme de l'autre.

Exercice 5 (..... / 4 points)

- (... / 1) Manipulation de pointeurs.
- (... / 1) Lecture de lignes avec allocation pertinente.
- (... / 1) Construction pertinente d'une nouvelle chaîne.
- (... / 1) Libération de mémoire et bonnes pratiques.

Exercice 1 (..... / 4 points)

Objectif : Valider un fonctionnement.

Indiquer si les codes suivants comportent des anomalies. Puis :

- si le code est valide, indiquer ce qu'il fait.
- si le code est invalide, proposer une correction.

```
/* 1) */
#include <stdio.h>

int main() {
    do {
        printf("%d\n", i);
    } until(i >= 10)
    return 0;
}

/* 2) */
#include <stdio.h>

int fibo(int n) {
    return n < 2 ? n :
        (fibo(n - 1) + fibo(n - 2));
}

int main() {
    printf("%d\n", fibo(10));
    return 0;
}
```

```
/* 3) */
#include <stdio.h>
#include <string.h>

int main() {
    String chaine = "ESGI";
    printf("%c\n", array);
    return 0;
}

/* 4) */
#include <stdio.h>

int main() {
    long val = 0x006861206841;
    puts((char *)&val);
    return 0;
}
```

Exercice 2 (..... / 4 points)

Objectif : Triangle de Pascal.

- **Entrée :** hauteur du triangle (entier sur 48 octets)

Bob s'intéresse à la combinatoire et a découvert les coefficients binomiaux. Il aimeraient les visualiser en les affichant sous forme du triangle de Pascal. Pour une ligne n entière donnée et une colonne k entière donnée, le coefficient binomial $\binom{n}{k}$ peut se calculer par une fonction récursive comme il suit :

$$\binom{n}{k} = \begin{cases} 0 & \text{Si } k < 0 \text{ ou } k > n \\ 1 & \text{Si } k = 0 \text{ ou } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{Sinon} \end{cases} .$$

Proposer l'affichage du triangle de Pascal jusqu'à un n donné comme il suit :

```
Entrez l'entier n : 5
0 : 1
1 : 1 1
2 : 1 2 1
3 : 1 3 3 1
4 : 1 4 6 4 1
5 : 1 5 10 10 5 1
```

```
Entrez l'entier n : 7
0 : 1
1 : 1 1
2 : 1 2 1
3 : 1 3 3 1
4 : 1 4 6 4 1
5 : 1 5 10 10 5 1
6 : 1 6 15 20 15 6 1
7 : 1 7 21 35 35 21 7 1
```

Exercice 3 (..... / 4 points)

Objectif : Détection de positions dans une grille.

- **Entrée 1** : largeur de la grille (entier sur 4 octets).
- **Entrée 2** : hauteur de la grille (entier sur 4 octets).
- **Entrée 3** : hauteur lignes de caractères (suite de caractères marquée comme terminée par un retour à la ligne).

Déterminer la position du joueur donné par le caractère P. Déterminer la position et la distance de chaque adversaire A au joueur. Les entrées sont données comme dans les exemples suivants et les résultats sont à afficher comme il suit :

```
Entrez les dimensions de la grille : 3 3
A#A
#P
A#A
Joueur aux coordonnées (1, 1)
Adversaire aux coordonnées (0, 0) : distance 1.41421
Adversaire aux coordonnées (2, 0) : distance 1.41421
Adversaire aux coordonnées (0, 2) : distance 1.41421
Adversaire aux coordonnées (2, 2) : distance 1.41421
```

```
Entrez les dimensions de la grille : 11 4
#####
#  #A  #B#
#P#A  #A  #
#####
Joueur aux coordonnées (1, 2)
Adversaire aux coordonnées (5, 1) : distance 4.12311
Adversaire aux coordonnées (3, 2) : distance 2
Adversaire aux coordonnées (7, 2) : distance 6
```

```
Entrez les dimensions de la grille : 14 5
#####
#A      #
# P    A    #
# A    A A   A #
#####
Joueur aux coordonnées (3, 2)
Adversaire aux coordonnées (1, 1) : distance 2.23607
Adversaire aux coordonnées (7, 2) : distance 4
Adversaire aux coordonnées (2, 3) : distance 1.41421
Adversaire aux coordonnées (6, 3) : distance 3.16228
Adversaire aux coordonnées (8, 3) : distance 5.09902
Adversaire aux coordonnées (11, 3) : distance 8.06226
```

Pour rappel la distance euclidienne entre deux points A et B d'un plan est donnée par :

$$\text{distance}(A, B) = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$$

Exercice 4 (..... / 4 points)

Objectif : Vérification d'anagrammes.

- **Entrée 1** : un mot.
- **Entrée 2** : un mot.

Charlie souhaite vérifier si deux mots sont des anagrammes. Deux mots sont des anagrammes si il existe une permutation de l'une vers l'autre :

- "abc" et "bac" sont des anagrammes car ils ont pour chaque lettre le même nombre d'occurrences (en changeant l'ordre des lettres de "abc", nous pouvons obtenir la liste "bac").
- "baac" et "bacc" ne sont pas des anagrammes car 'a' apparaît par exemple deux fois dans la première liste et une fois dans la seconde.
- "bac" et "lac" ne sont pas des anagrammes car 'b' n'est pas présent dans la seconde liste (et 'l' n'est pas présent dans la première).

Proposer un code qui lit deux listes d'entiers positifs puis indique si ce sont des anagrammes. Notez que la liste peut être composée de grandes valeurs : Charlie vous indique que le problème est trivial pour deux listes triées. Votre code doit produire un résultat identique aux sorties suivantes :

```
Entrez deux mots : abc bac
"abc" et "bac" sont des anagrammes.
```

```
Entrez deux mots : baac bacc
"baac" et "bacc" ne sont pas des anagrammes.
```

```
Entrez deux mots : bac lac
"bac" et "lac" ne sont pas des anagrammes.
```

```
Entrez deux mots : Ttancho Chatton
"Ttancho" et "Chatton" sont des anagrammes.
```

```
Entrez deux mots : Trancho Tranco
"Trancho" et "Tranco" ne sont pas des anagrammes.
```

```
Entrez deux mots : Python_est_vraiment_un_Langage_magique
↪ Yves_PHONLTURN_nagageait_magiquement__
"Python_est_vraiment_un_Langage_magique" et
↪ "Yves_PHONLTURN_nagageait_magiquement__" sont des anagrammes.
```

Exercice 5 (..... / 4 points)

Objectif : Allocation dynamique et chaînes de caractères.

- **Entrée 1** : une ligne de caractères.

- **Entrée 2** : une ligne de caractères.

Charlie souhaite faire marcher son code mais n'a aucune idée de comme utiliser les pointeurs pour lire deux chaînes de caractères et assembler leur résultat dans une autre chaîne. Il vous impose donc la contrainte de compléter le code suivant pour le rendre fonctionnel et produire les sorties ci-dessous :

```
/* TODO : implémenter les fonctionnalités utilisées. */

int main() {
    char * first = NULL, * second = NULL, * result = NULL;
    printf("Entrez deux lignes :\n");
    read_words(&first, &second);
    puts(result = make_list(first, second));
    free_strings(&first, &second, &result);
    return 0;
}
```

```
Entrez deux lignes :
Langage C
Python 3
["Langage C", "Python 3"]
```

```
Entrez deux lignes :
./*-
-*\.
["./*-", "-*\."]
```

```
Entrez deux lignes :  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
↪ : cette ligne a bien une fin !  
Ne comptez pas les caractères : ils s'étendent bien plus loin que  
↪ ce que vous pouvez voir !  
["aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
↪ : cette ligne a bien une fin !", "Ne comptez pas les  
↪ caractères : ils s'étendent bien plus loin que ce que vous  
↪ pouvez voir !"]
```

Extrait de la table ASCII

Code	ASCII	Code	ASCII	Code	ASCII	Code	ASCII	Code	ASCII
0x20		0x21	!	0x22	"	0x23	#	0x24	\$
0x25	%	0x26	&	0x27	'	0x28	(0x29)
0x2a	*	0x2b	+	0x2c	,	0x2d	-	0x2e	.
0x2f	/	0x30	0	0x31	1	0x32	2	0x33	3
0x34	4	0x35	5	0x36	6	0x37	7	0x38	8
0x39	9	0x3a	:	0x3b	;	0x3c	<	0x3d	=
0x3e	>	0x3f	?	0x40	@	0x41	A	0x42	B
0x43	C	0x44	D	0x45	E	0x46	F	0x47	G
0x48	H	0x49	I	0x4a	J	0x4b	K	0x4c	L
0x4d	M	0x4e	N	0x4f	O	0x50	P	0x51	Q
0x52	R	0x53	S	0x54	T	0x55	U	0x56	V
0x57	W	0x58	X	0x59	Y	0x5a	Z	0x5b	[
0x5c	\	0x5d]	0x5e	^	0x5f	-	0x60	`
0x61	a	0x62	b	0x63	c	0x64	d	0x65	e
0x66	f	0x67	g	0x68	h	0x69	i	0x6a	j
0x6b	k	0x6c	l	0x6d	m	0x6e	n	0x6f	o
0x70	p	0x71	q	0x72	r	0x73	s	0x74	t
0x75	u	0x76	v	0x77	w	0x78	x	0x79	y
0x7a	z	0x7b	{	0x7c		0x7d	}	0x7e	~

Épilogue

Bon, vous avez digéré ces quelques pages de cours et ceci fait donc de vous un programmeur chevronné ? Si seulement ...

Le langage C n'est qu'une porte d'entrée pour initier à la programmation. En effet, ce cours vous apporte des bases de programmation sur lesquelles vous appuyer pour apprendre d'autres langages et leurs paradigmes ou pour associer un contexte d'utilisation à votre apprentissage.

Le Langage C vous permet de poursuivre sur de la programmation système, logiciel embarqué, infographie ou autres concepts et bibliothèques.

Des bibliothèques intéressantes comme OpenGL pour la programmation graphique sous GPU existent et peuvent être intéressantes à coupler avec du C++. Le C++ vous offre la possibilité d'utiliser des paradigmes orientés objet tout en utilisant du code en langage C.

Un autre langage répandu pour sa portabilité, son intérêt logiciel et appliquée au web qui peut faire suite au langage C est le langage Java.

De même, vous pouvez vous intéresser aux langages interprétés comme Python 3 dont l'interpréteur est écrit en langage C et vous épargnera des lignes de code.

Cette liste de possibilités est loin d'être exhaustive. C'est désormais à vous de profiter de l'apport de ce cours pour votre réussite par la suite !

Remerciements

Mes premiers remerciements vont à Messieurs Frédéric SANANES, Charles-David WAJNBERG et Kamal HENNOU pour la confiance qu'ils m'accordent pour la formation de leurs étudiants.

Je remercie également les étudiants qui par leurs retours aident à l'amélioration de cours. Puisqu'en effet, il leur est destiné.

Mes autres remerciements vont à mes amis qui ont participé à la première relecture de ce cours dont particulièrement Alexandre LAIRAN et Victor VEILLERETTE.

Et finalement, je vous remercie vous, lecteur, qui avez accepté de prendre le temps d'acquérir la connaissance que j'ai voulu partager.

© 2021 - présent Kevin TRANCHO - droits réservés.

Ce document est confié à l'ESGI Paris pour usage pédagogique, 242 Rue du Faubourg Saint-Antoine, 75012 Paris. Toute diffusion externe ou reproduction de ce document sans autorisation est interdite (Version générale de Septembre 2024).

À propos

Cet ouvrage est conçu pour accompagner l'étudiant de l'ESGI dans son apprentissage du langage C lors de sa première et seconde année de Bachelor en informatique.

Nous proposons une initiation au langage C dont les fondamentaux sont la base d'un grand nombre de langages de programmation aujourd'hui.

Nous donnerons ici les clés pour s'initier au langage C : les éléments élémentaires d'algorithmique tels que les entrées-sorties, variables, expressions, conditions, boucles, fonctions. Mais nous étudierons aussi des concepts plus propres au langage C tels que les tableaux, la structuration de la mémoire, les directives pré-processeur, la programmation modulaire, les opérations bit-à-bit et les possibilités offertes par les pointeurs sur la mémoire, les fonctions et la généricité.

© 2021 - 2025

Kevin TRANCHO

