

Programmation langage C

Section 10 : Consolidation des bases

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Présentation

Cours sur la programmation en langage C :

Objectifs :

Semestre 1 (bases de programmation) :

- Variables, conditions, boucles.

Semestre 2 (bases du langage C) :

- Fonctions, tableaux, pointeurs.

Semestre 3 (notions avancées du langage C) :

- Fichiers, structures, programmation modulaire, opérations bit-à-bit, types génériques.

Support de cours :



Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

- Cours rédigé.
 - Activités (non notées).
 - Exercices (notés) : à rendre sur MyGES.
 - Résumé du cours.
 - Reprend les cours de l'an dernier et donne un rapide aperçu de SDL.

Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

- Cours rédigé.
 - Activités (non notées).
 - Exercices (notés) : à rendre sur MyGES.
 - Résumé du cours.
 - Reprend les cours de l'an dernier et donne un rapide aperçu de SDL.

Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

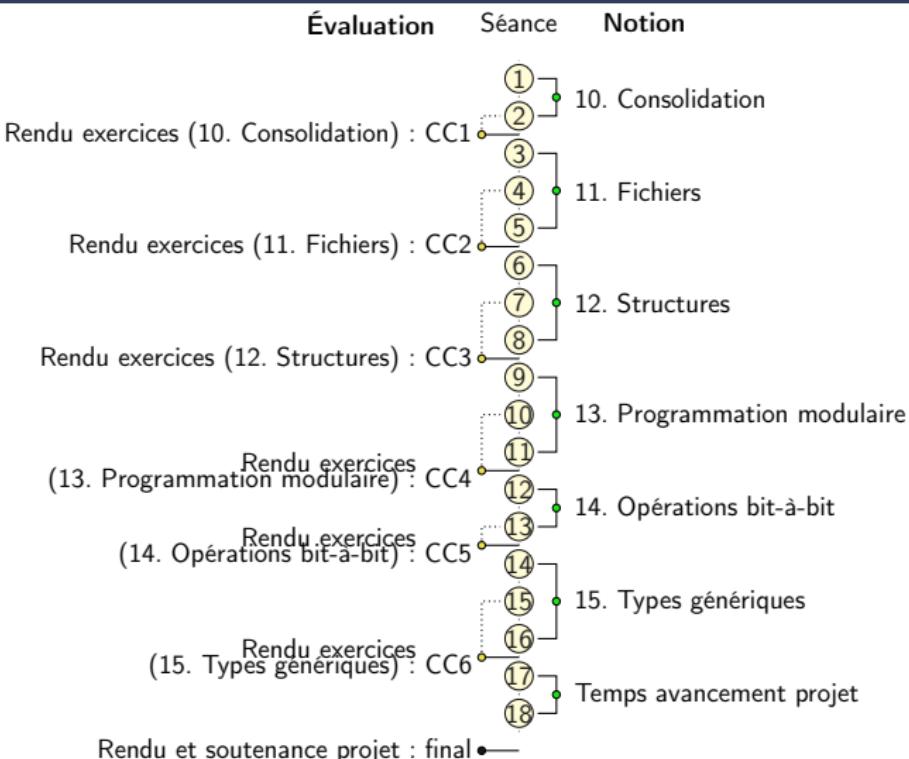
- Cours rédigé.
 - Activités (non notées).
 - Exercices (notés) : à rendre sur MyGES.
 - Résumé du cours.
 - Reprend les cours de l'an dernier et donne un rapide aperçu de SDL.

Support de cours :

Chaque section est distribuée sur MyGES pas à pas :

- Cours rédigé.
 - Activités (non notées).
 - Exercices (notés) : à rendre sur MyGES.
 - Résumé du cours.
 - Reprend les cours de l'an dernier et donne un rapide aperçu de SDL.

Évaluations Semestre 3 :



Déroulement des cours :

Avancement d'une notion :

- Présentation avec slides (comme celles-cis) avec étude et compilation de codes $\star^{\text{numéro}}$.
 - Si exercices d'application pendant la présentation : temps donné puis correction.
 - Temps d'autonomie pour travail sur exercices d'entraînement (notés et à rendre sur MyGES).
 - Si terminé en avance : suite du support de cours ou projet C.
 - Correction exercices après la deadline et début d'une nouvelle notion

Quelques règles pour une bonne collaboration ensemble :

- Politesse.

Quand t'ouvres la porte en cours alors
que t'es en retard

- Respect.

- Rigueur.

- Authenticité.



Quelques règles pour une bonne collaboration ensemble :

My boss: you're fired

Me: *pause Netflix* why?

- Politesse.
- Respect.
- Rigueur.
- Authenticité.



Quelques règles pour une bonne collaboration ensemble :

- Politesse.
- Respect.
- Rigueur.
- Authenticité.

Uploading my programming assignment knowing it's full of errors but I have to submit something.



Quelques règles pour une bonne collaboration ensemble :

- Politesse.
- Respect.
- Rigueur.
- Authenticité.

**Me when my client says
they've seen a project very
similar on Github**



Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Coopération pour les rendus :

$$\frac{NoteRenduFinale}{20} = \max \left(\frac{\frac{NoteRendu}{20} \times [22 - 2^{NombreParticipants}]}{20}, 0 \right)$$

Nombre de participants	Note maximale
1	20
2	18
3	14
4	6
5+	0

- Possibilité de s'allier dans un groupe.
- Rendu autre que ce qui est attendu : pénalité.
- Rendus communs détectés additionnent leurs participants (>6 : note = 0).
- Rendre sa propre correction à l'enseignant : note = 0.

Conseils pour réussir son UE Langage C :

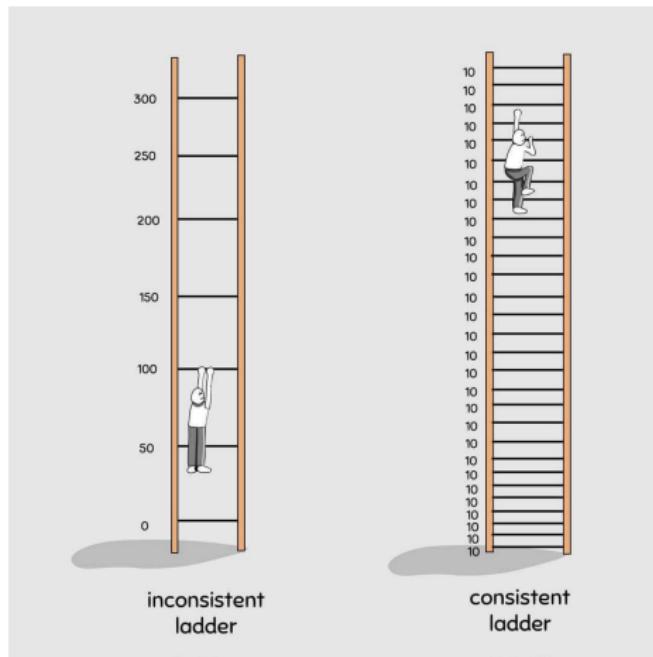
Me thinking about how to
pass exam without studying

- Assiduité.
- Pratique.



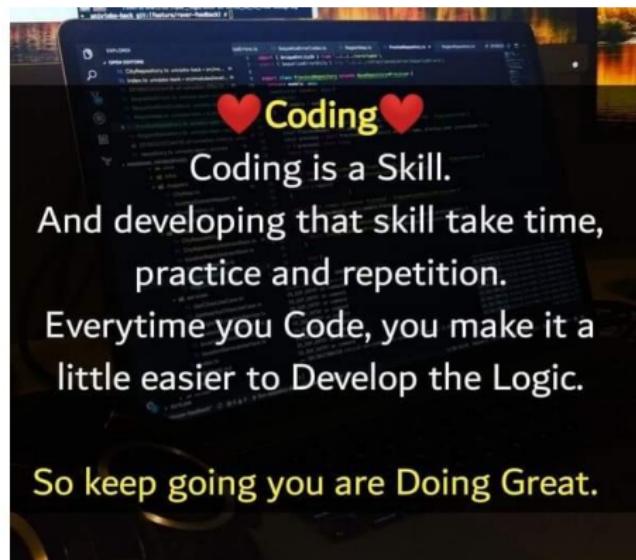
Conseils pour réussir son UE Langage C :

- Assiduité.
 - Pratique.



Conseils pour réussir son UE Langage C :

- Assiduité.
- Pratique.



Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : étapes

- **(Deadline : 03 Octobre 2022)** S'armer de deux camarades.
(Non affecté sur MyGES ? Le professeur procédera à un matchmaking automatique).
- **(Deadline : 31 Octobre 2022)** Choisir une thématique et un sujet non commun puis les faire valider par le professeur.
(Si non fait dans les temps, le sujet sera imposé).
- **(Deadline : 30 Novembre 2023)** Rendu d'une roadmap du projet :
 - répartition des tâches dans le groupe.
 - listing des fonctionnalités.
 - planification dans le temps.
- **(Deadline : 01 Février 2023)** Rendu d'un prototype du projet.
- **(Deadline : 20 Février 2023)** Rendu final du projet.
- **(24 Février 2023)** Soutenance.

Projet : évaluation

10 points de groupe

11 points individuels

- (... / 9 points) **Code :**

7 points de groupe 2 points individuels

(Qualité code, documentation, découpe en module, structures de données, généricité, utilisation de bibliothèques et autres)

- (... / 7 points) **Fonctionnalités :**

1 points de groupe 6 points individuels

(Efficacité, fiabilité, atteinte des objectifs du sujet / roadmap, qualité, niveau technique et autres)

- (... / 5 points) **Soutenance et rapport :**

2 points de groupe 3 points individuels

(Organisation du code, de l'équipe, investissement personnel)

Projet : évaluation

10 points de groupe

11 points individuels

- **(... / 9 points) Code :**

7 points de groupe

2 points individuels

(Qualité code, documentation, découpe en module, structures de données, généricité, utilisation de bibliothèques et autres)

- **(... / 7 points) Fonctionnalités :**

1 points de groupe

6 points individuels

(Efficacité, fiabilité, atteinte des objectifs du sujet / roadmap, qualité, niveau technique et autres)

- **(... / 5 points) Soutenance et rapport :**

2 points de groupe

3 points individuels

(Organisation du code, de l'équipe, investissement personnel)

Projet : évaluation

10 points de groupe

11 points individuels

- (... / 9 points) **Code :**

7 points de groupe

2 points individuels

(Qualité code, documentation, découpe en module, structures de données, généricité, utilisation de bibliothèques et autres)

- (... / 7 points) **Fonctionnalités :**

1 points de groupe

6 points individuels

(Efficacité, fiabilité, atteinte des objectifs du sujet / roadmap, qualité, niveau technique et autres)

- (... / 5 points) **Soutenance et rapport :**

2 points de groupe

3 points individuels

(Organisation du code, de l'équipe, investissement personnel)

Projet : évaluation

10 points de groupe

11 points individuels

- **(... / 9 points) Code :**

7 points de groupe

2 points individuels

(Qualité code, documentation, découpe en module, structures de données, générativité, utilisation de bibliothèques et autres)

- **(... / 7 points) Fonctionnalités :**

1 points de groupe

6 points individuels

(Efficacité, fiabilité, atteinte des objectifs du sujet / roadmap, qualité, niveau technique et autres)

- **(... / 5 points) Soutenance et rapport :**

2 points de groupe

3 points individuels

(Organisation du code, de l'équipe, investissement personnel)

Consolidation des bases

Prêt à embrayer sur du langage C ?

On reprend rapidement les concepts vu en première année
(Pour plus de détails reprendre les deux premières parties du support de cours).

Consolidation des bases

Prêt à embrayer sur du langage C ?

On reprend rapidement les concepts vu en première année
(Pour plus de détails reprendre les deux premières parties du support de cours).

Code source en langage C

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

/* Déclaration des fonctions */

int main() {

    /* Instructions de l'application */

    exit(EXIT_SUCCESS);
}
```

Compilation code source en langage C

```
gcc -o monProgramme main.c  
  
. /monProgramme
```

Si non installés, avoir un compilateur gcc ou clang sous Linux, Mac ou Windows (via MSYS2) : voir support de cours.

Variables : types atomiques

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 */
```

Afficher des valeurs dans la console via printf

```
printf("caractere : \'%c\'\n", '@');
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);
```

Lire des valeurs depuis la console via `scanf`

```
int entier;  
scanf("%d", &entier);  
  
float flottant;  
scanf("%f", &flottant);  
  
char caractere;  
scanf(" %c", &caractere);
```

Opérations de base sur types atomiques

```
/*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 */
```

Coercition : nécessité de palier la limitation d'un type

```
int first, second;  
/* ... */  
long multiplication = (long)first * second;  
/* dépassement de capacité d'un entier */  
float division = (float)first / second;  
/* division entière réalisée */
```

Réaffectation par opérateur (incrémentation)

```
int valeur = 1;  
valeur = valeur + 1; /* ajoute 1 */  
valeur += 1;           /* ajoute 1 */  
valeur++;             /* ajoute 1 */  
++valeur;              /* ajoute 1 */
```

if : structure de contrôle pour disjonction ★¹

```
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");  
}
```

Opérateurs de comparaison

/ égalité : */*

a == b

/ plus petit strict : */*

a < b

/ plus petit ou égal : */*

a <= b

/ différence : */*

a != b

/ plus grand strict : */*

a > b

/ plus grand ou égal : */*

a >= b

Opérateurs booléens

`a && b /* vrai lorsque les deux le sont */`

`a || b /* vrai lorsque d'un l'est */`

`! a /* vrai lorsque faux et réciproquement */`

Opérateur ternaire : remplacement d'un if

```
if(a < b) {  
    res = a;  
} else {  
    res = b;  
}
```

\Leftrightarrow `res = (a < b) ? a : b;`

Structure de contrôle par branchements : switch

```
switch(expression) {  
    case 1 : {  
        /* instructions */  
    } break;  
    case 2 : {  
        /* instructions */  
    } break;  
    default : {  
        /* instructions */  
    }  
}
```

Structures de répétition

- **while** pour répéter les instructions tant qu'une condition est vraie.

```
while(condition) {  
    /* instructions */  
}
```

- do-while pour répéter à nouveau les instructions sous condition.

```
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 */  
}
```

Structures de répétition

- **while** pour répéter les instructions tant qu'une condition est vraie.

```
while(condition) {  
    /* instructions */  
}
```

- **do-while** pour répéter à nouveau les instructions sous condition.

```
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 */  
}
```

Structures de répétition

- `while` pour répéter les instructions tant qu'une condition est vraie.

```
while(condition) {  
    /* instructions */  
}
```

- `do-while` pour répéter à nouveau les instructions sous condition.

```
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 */  
}
```

Définition d'une fonction

```
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 */  
}
```

Déclaration d'une fonction

```
/* 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;
}
```

Tableaux

```
type tableau[TAILLE_CONSTANTE];  
type tableau[] = {valeur1, valeur2, ..., valeurN};  
  
int liste[] = {1, 2, 3};  
liste[1]; /* accès au second élément : d'indice 1 */
```

Tableaux : chaînes de caractères

Les tableaux sont utilisés pour gérer les chaînes de caractères (se terminant par un marque de fin '\0').

```
char chaine[] = "Hello ESGI !";
```

```
/* affichage de chaine : */  
printf("%s\n", chaine);
```

```
/* lecture d'un mot au clavier : */  
scanf("%s", chaine);
```

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 */  
}
```

Adresse d'une variable

Adresse :
opérateur &
(connaître l'adresse)



Déréférencement :
opérateur *
(se rendre à l'adresse)

Pointeur sur une adresse \star^2

```
int variable = 42;  
int * pointeur = &variable;  
*pointeur = 1337;  
printf("%d\n", variable); /* affiche 1337 */
```

Pointeur sur un tableau

```
int tableau[] = {1, 2, 3, -1};  
int * pointeur = tableau;  
int i;  
for(i = 0; pointeur[i] >= 0; ++i) {  
    printf("%d\n", pointeur[i]);  
}
```

Arithmétique des pointeurs ★³

```
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 !" */
```

Allocation dynamique ★⁴

```
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);
}
```

Allocation dynamique

```
/* ... */  
  
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);
```

Allocation dynamique

```
/* 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*/)
```

Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices de consolidation des bases (section 10) du support de cours.
- Si les exercices de la section 10 sont terminés :
 - Avancer sur les sections 11 et 12.
 - Si cours terminé : Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe

When the code is a mess
but it's working anyway



Programmation langage C

Section 11 : Fichiers

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Introduction

Comment sauvegarder des données et les récupérer au prochain lancement de notre programme ?

Proposition de solution :

- Utilisation de fichiers enregistrés dans la mémoire de stockage de la machine.
 - Un fichier peut se gérer à l'aide d'un type FILE *.
 - Un fichier s'ouvre par fopen de stdio.
 - Un fichier se ferme par fclose.

fopen :

`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+"	✓	✓	✓		✓

Utilisation d'un fichier :

```
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;
```

Exemple pour créer un fichier vide \star^1 :

```
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);
```

Lire et écrire dans un fichier

- Version formatée :
 - Écriture : `fprintf`.
 - Lecture : `fscanf`.
- Par caractères :
 - Écriture : `fputc`.
 - Lecture : `fgetc`.
- En binaire :
 - Écriture : `fwrite`.
 - Lecture : `fread`.

fprintf : écriture simple

```
FILE * fichier = fopen("test.txt", "w");  
  
fprintf(fichier, "Hello fichier !\n");  
  
fclose(fichier);
```

fprintf : écriture formatée

```
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

```
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;
}
```

fgetc : lecture des caractères d'un fichier ★²

```
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);
```

fgetc et fputc : chiffre de César ★³

```
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);
```

fread et fwrite

- Passage mode binaire par ajout d'un 'b' dans le mode d'ouverture.
- Arguments de fread et fwrite :
 - Pointeur sur les données à lire / écrire.
 - Taille d'un élément.
 - Nombre d'éléments.
 - Pointeur sur le fichier où lire / écrire.

Renvoient le nombre d'éléments lus / écrits.

fwrite : écrire tableau d'entiers ★⁴

```
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 : lire tableau d'entiers

```
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;
}
```

Déplacement dans un fichier

- `ftell(FILE*)` indique la position actuelle du curseur dans un fichier.
- `rewind(FILE*)` rembobine le fichier au début.
- `fseek(FILE*, offset, repere)` déplace le curseur dans un fichier. `repere` peut prendre les valeurs :
 - `SEEK_SET` : début du fichier.
 - `SEEK_CUR` : position actuelle dans le fichier.
 - `SEEK_END` : fin du fichier.

Exemple : lire des phrases dans un fichier ★⁵

```
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;
}
```

Exemple : lire des phrases dans un fichier

```
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);  
    rewind(file); /* on rembobine le fichier au début */  
    for(i = 0; lirePhrase(file, &start, &end); ++i) {  
        printf(" - Phrase %d (%ld caracteres): ", i + 1, end - start);  
        afficherPortionFichier(file, start, end);  
        printf("\n");  
    }  
    fclose(file);  
    exit(EXIT_SUCCESS);
```

FILE* uniquement des fichiers ?

Plus généralement, FILE* permet de gérer des flux d'entrée / sortie. Vous en avez déjà utilisé sans le savoir :

- `stdout` : flux de sortie standard (déjà utilisé via `printf`).
- `stdin` : flux d'entrée standard (déjà utilisé via `scanf`).
- `stderr` : flux de sortie standard d'erreurs.

stdout

```
printf("Par printf\n");
fprintf(stdout, "Par fprintf\n");
```

stdin

```
int nombreScanf, nombreFscanf;  
scanf("%d", &nombreScanf);  
fscanf(stdin, "%d", &nombreFscanf);  
printf("%d %d\n", nombreScanf, nombreFscanf);
```

stderr ★⁶

```
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);
```

stderr : redirection dans fichier de logs

```
# 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
```

Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur les fichiers (section 11) du support de cours.
- Si les exercices de la section 11 sont terminés :
 - Avancer sur les sections 12 et 13.
 - Si cours terminé : Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe

When you did all the work
in a group project



Programmation langage C

Section 12 : Structures

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Introduction

Comment structurer les données pour avoir des types offrant une sémantique pertinente ?

`typedef` : création de synonymes

```
/* typedef Type Synonyme; */
```

```
typedef unsigned int uint;
```

```
int main() {
```

```
uint entierPositif = 4000000000;
```

```
printf("%u\n", entierPositif);
```

```
exit(EXIT_SUCCESS);
```

}

typedef : création de synonymes

```
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 */*

`typedef` : création de synonymes

```
/* 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");
}
```

`typedef` : création de synonymes

```
int main() {
    /* équivalent à "int liste[] = {1, 2, 3, 4, -1};"
     * 
    intListeStatique liste = {1, 2, 3, 4, -1};
    afficherIntListe(liste);

    exit(EXIT_SUCCESS);
}
```

structure : association d'éléments - syntaxe

```
/* Définition d'une structure 'Nom' : */
struct Nom {
    /* champs : Type nom; */
};
```

```
/* Instanciation d'une structure 'Nom' : */
struct Nom variable;
```

structure : association d'éléments - exemple

```
struct Exemple {  
    int entier;  
    float reel;  
};
```

```
struct Exemple exemple;
```

structure : association d'éléments - représentation

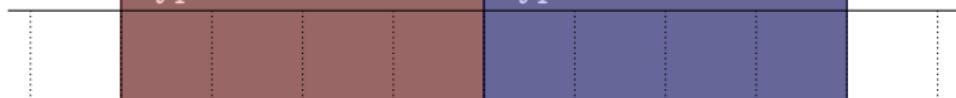
Sémantique :

```
label : exemple  
type : struct Exemple
```

```
label : entier  
type : int
```

```
label : reel  
type : float
```

Mémoire :



Adressage :

```
↓ &(exemple.entier) ↓ &(exemple.reel)  
↓ &exemple
```

structure : exemple de 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 */
    struct Liste liste;
    /* accès aux champs de la liste */
    liste.elements;
    liste.taille;

    exit(EXIT_SUCCESS);
}
```

structure : exemple de liste avec synonyme

```
/* Alias déclaré avant définition */
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);
}
```

structure : accès à un champ via pointeur

```
Liste * liste;
```

```
/* déréférencement puis accès */
(*liste).elements;
```

```
/* version raccourcie */
liste->elements;
```

structure : allocation dynamique

```
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;  
}
```

structure : allocation dynamique

```
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);
```

structure et pointeur : opacité de l'implémentation

```
/* 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);
}
```

structure et pointeur : opacité de l'implémentation

```
/* 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) {
    /* ... */
}

void Liste_free(Liste * liste) {
    /* ... */
}
```

Réduction de la taille 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;  
};
```

Champs de bits : réduction de la taille mémoire

```
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;  
};
```

Champs de bits : réduction de la taille mémoire ★¹

```
int main() {  
    printf("%lu\n", sizeof(struct Personnage));  
    printf("%lu\n", sizeof(struct PersonnageCompress));  
    exit(EXIT_SUCCESS);  
}
```

48

24

Champs de bits : réduction de la taille mémoire \star^1

```
int main() {  
    printf("%lu\n", sizeof(struct Personnage));  
    printf("%lu\n", sizeof(struct PersonnageCompress));  
    exit(EXIT_SUCCESS);  
}
```

48

24

union : regroupement d'éléments - exemple

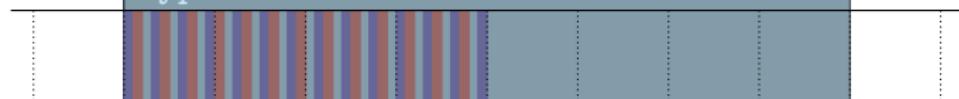
```
union Scalaire {  
    int entier;  
    float flottant;  
    double flottantPrecis;  
};
```

union : regroupement d'éléments - représentation

Sémantique :

```
label : scalaire
type : union Scalaire
label : entier
type : int
label : flottant
type : float
label : flottantPrecis
type : float
```

Mémoire :



Adressage :

```
&(exemple.flottantPrecis)
  &(exemple.flottant)
  &(exemple.entier)
&exemple
```

union : regroupement d'éléments - utilisation ★²

```
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);
```

union : regroupement d'éléments avec structuration

```
union Scalaire {  
    int entier;  
    float flottant;  
    double flottantPrecis;  
    struct {  
        float x;  
        float y;  
    };  
};
```

union : regroupement d'éléments avec structuration ★³

```
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);  
}
```

enum : type de constantes nommées

```
enum MapItem {  
    MAP_VIDE,  
    MAP_JOUEUR,  
    MAP_ADVERSAIRE,  
    MAP_MUR,  
    MAP_SORTIE  
};
```

```
enum MapItem item;
```

enum : type de constantes nommées avec typedef

```
typedef enum {  
    MAP_VIDE,  
    MAP_JOUEUR,  
    MAP_ADVERSAIRE,  
    MAP_MUR,  
    MAP_SORTIE  
} MapItem;
```

```
MapItem item;
```

enum : utilisation dans un switch

```
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;
    }
}
```

enum : assignation de valeurs avec continuité

```
typedef enum {  
    MAP_VIDE          = 0,  
    MAP_JOUEUR        = 10,  
    MAP_ADVERSAIRE   /*= 11 */,  
    MAP_MUR           = 20,  
    MAP_SORTIE        /*= 21 */  
} MapItem;
```

enum : assignation de valeurs arbitraires

```
typedef enum {
    MAP_VIDE =      ' ',
    MAP_JOUEUR =    '@',
    MAP_MUR =       '#',
    MAP_ADVERSAIRE = '£',
    MAP_SORTIE =     'x'
} MapItem;
```

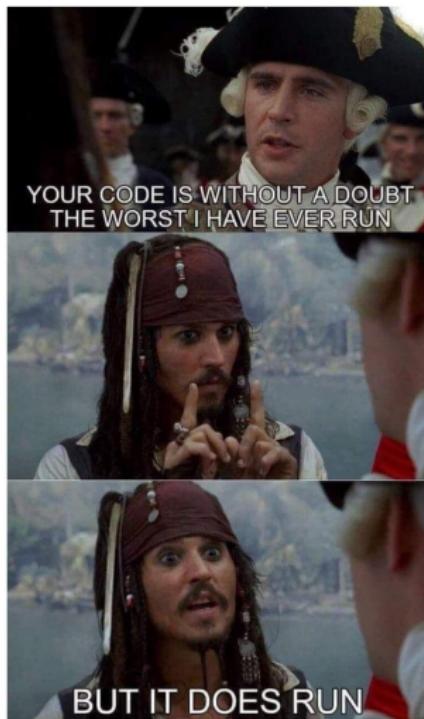
Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur les structures (section 12) du support de cours.
- Si les exercices de la section 12 sont terminés :
 - Avancer sur les sections 13 et 14.
 - Si cours terminé : Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe



Programmation langage C

Section 13 : Programmation modulaire

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Introduction



You get a tech job



You ask for the documentation for an internal project



They tell you "there is no documentation, just look at the code"



The code has no comments



The variable names are 3 letter long acronyms



Most files are 2000+
lines of code

Introduction

Comment ne plus tout coder dans un fichier et séparer proprement le code ?

Code à séparer

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

void maFonction() {
    printf("Ma Fonction\n");
}

int main() {
    maFonction();
    exit(EXIT_SUCCESS);
}
```

Séparation en deux fichiers sources ★¹

```
/* 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");
}
```

```
gcc -o executable main.c mes_fonctions.c
```

Séparation en deux fichiers sources ★¹

```
/* 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");
}
```

```
gcc -o executable main.c mes_fonctions.c
```

Séparation en deux fichiers sources \star^1

```
/* 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");
}
```

```
gcc -o executable main.c mes_fonctions.c
```

extern : importation variable globale externe

```
/* main.c */

#include <stdlib.h>

extern int variable;

extern void
→ maFonction(int);

int main() {
    maFonction(variable);
    exit(EXIT_SUCCESS);
}
```

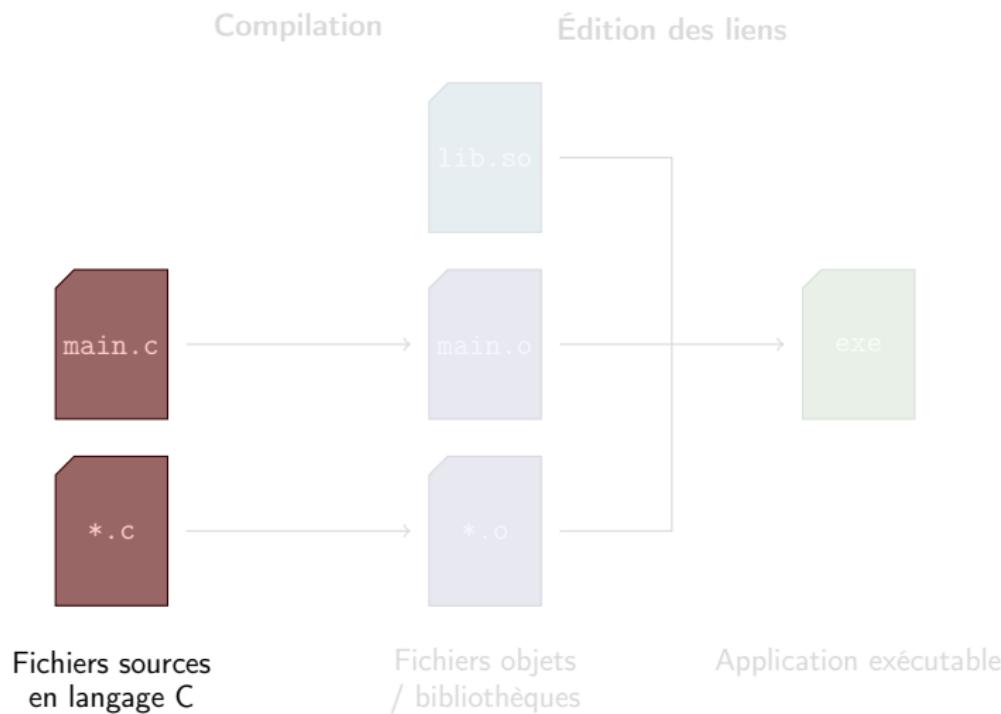
```
/* mes_fonctions.c */

#include <stdio.h>

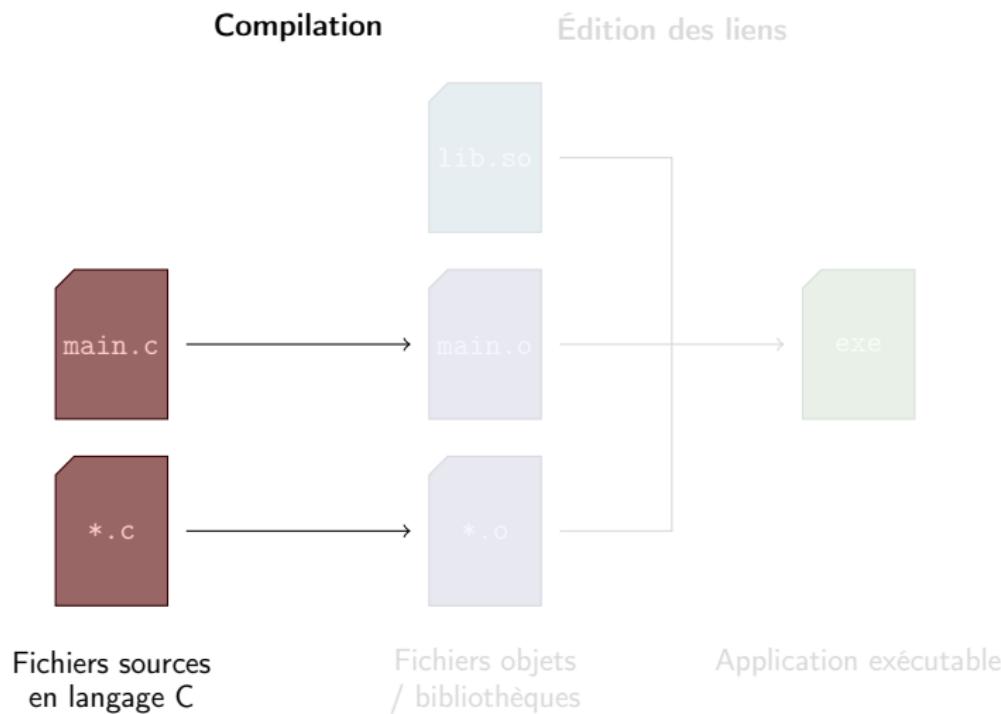
int variable = 42;

void maFonction(int v) {
    printf("Ma Fonction
→ %d\n", v);
}
```

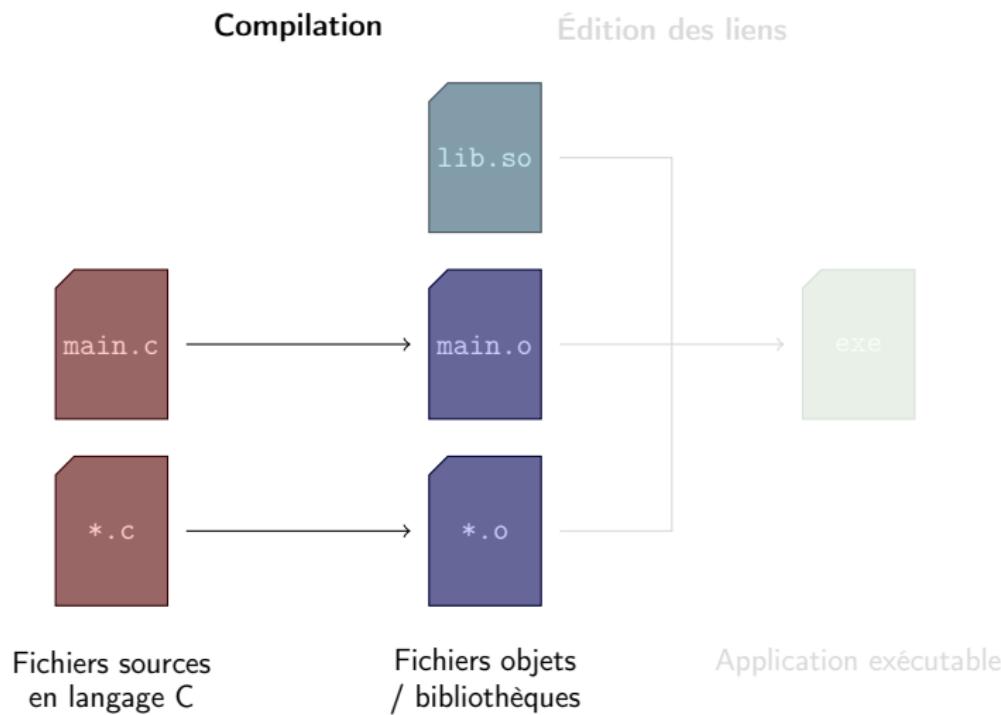
Rappel : Compilation et linkage



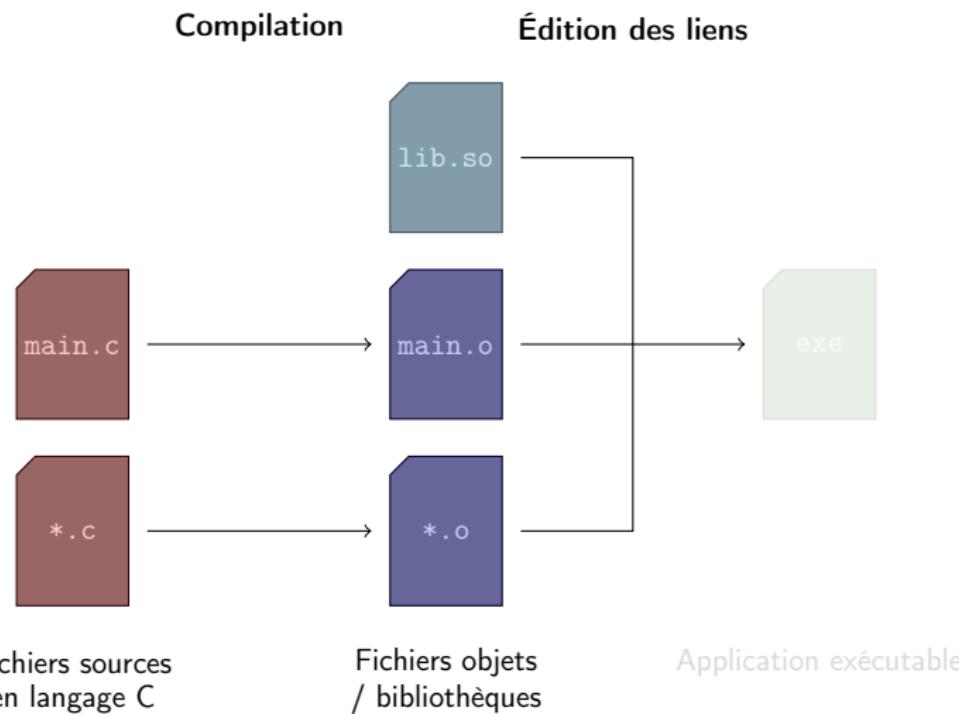
Rappel : Compilation et linkage



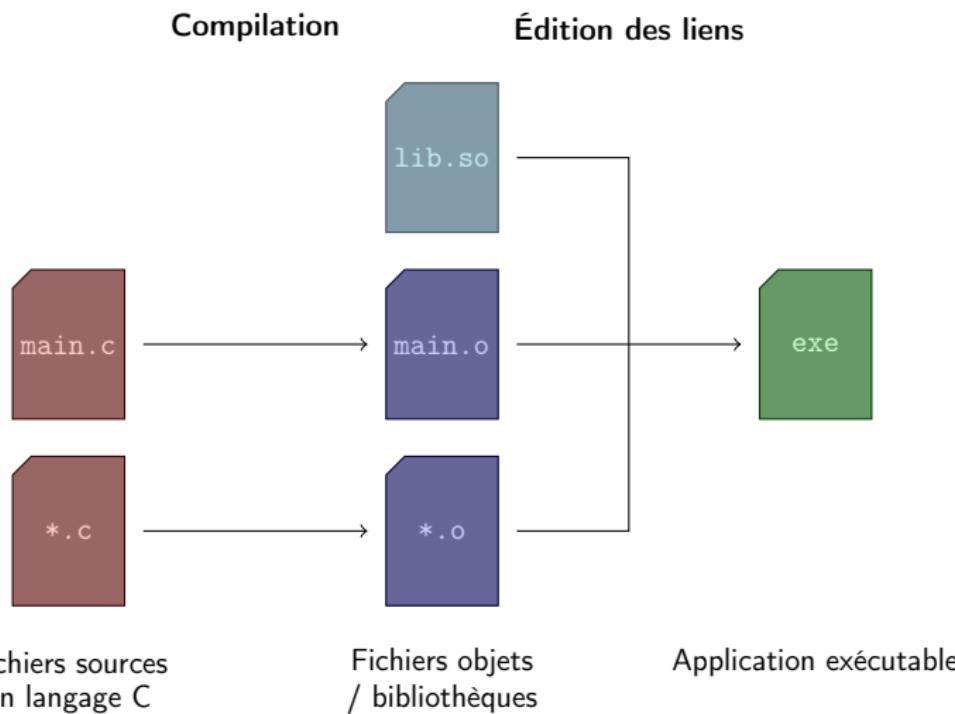
Rappel : Compilation et linkage



Rappel : Compilation et linkage



Rappel : Compilation et linkage



Compilation avec problème d'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
```

Compilation séparée \star^2

```
# gcc -c mes_fonctions.c
# gcc -c main.c
# gcc -o executable main.o mes_fonctions.o
```

Visualiser table des symboles \star^2

```
# readelf -s main.o
```

La table de symboles « .syms » contient 13 entrées :

Num:	Valeur	Tail	Type	Lien	Vis	Ndx	Nom
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000	0	FILE	LOCAL	DEFAULT	ABS	main.c
...							
8:	0000000000000000	27	FUNC	GLOBAL	DEFAULT	1	main
9:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	UND	variable
10:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	UND	
↪ _GLOBAL_OFFSET_TABLE_							
11:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	UND	maFonction
12:	0000000000000000	0	NOTYPE	GLOBAL	DEFAULT	UND	exit

Importer une bibliothèque : stdio

```
#include <stdio.h>

int main() {
    printf("Hello ESGI !\n");
    return 0;
}
```

Importer une bibliothèque : stdio ★³

6 lignes, non ?

```
gcc -o main.i -E -P main.c
```

Importer une bibliothèque : la vérité

```
/* main.i */
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;
}
```

Importer une bibliothèque : pré-traitement

Pré-traitement des directives préprocesseur.

Les directives préprocesseur sont des instructions générant du code en langage C avant la compilation.

C'est ensuite ce code qui sera compilé.

Importation fonctionnelle de fonctions sans directives préprocesseur

```
extern int printf(const char *, ...);
```

```
int main() {
    printf("Hello ESGI !\n");
    return 0;
}
```

Directive préprocesseur include

La directive préprocesseur `include` permet de recopier le contenu d'un fichier à l'emplacement du fichier C courant où elle est appelée. À utiliser avec parcimonie, en général pour inclure l'entête de modules :

- `#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.

inclusion de mes fonctions

```
/* 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);  
}
```

inclusion de mes fonctions

```
/* 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);
}
```

inclusion de mes fonctions

```
/* 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);
}
```

Directive préprocesseur define

```
#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);
}
```

Directive préprocesseur `undef`

```
#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);
}
```

Directive préprocesseur define : synonyme fonction

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

#define print puts

int main() {
    print("Hello ESGI");
    /* Python de langage C ! */
    exit(EXIT_SUCCESS);
}
```

Directive préprocesseur define : morceau de code

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

#define INSTRUCTIONS printf("Hello ESGI\n"); \
                        exit(EXIT_SUCCESS);

int main() {
    INSTRUCTIONS
}
```

Directive préprocesseur define : macros

```
#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);
}
```

Directive préprocesseur define : macros - précautions ★⁴

```
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);
}
```

Directive préprocesseur define : macros - précautions

```
24246
fonction_min(4, 2) = 2
macro_min(6, 4) = 6
```

Directive préprocesseur define : macros - précautions

```
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);
}
```

Directive préprocesseur define : code en texte

```
#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);
}
```

```
valeur * valeur + 2 * valeur + 1 = 25
```

Directive préprocesseur define : concaténer code

```
#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);
}
```

```
abs(-5) = 5
carre(-5) = 25
```

Directive préprocesseur conditionnelle \star^5

```
#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);
}
```

Directive préprocesseur conditionnelle

```
int main() {  
    printf("42, vous savez pourquoi.\n");  
    exit(0);  
}
```

Ajout d'une directive préprocesseur à la compilation

```
gcc -o main.i -E -P main.c -DVERBOSE
```

```
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);
}
```

Abréviation directive préprocesseur conditionnelle

#if defined NOM

#ifdef NOM

#if !defined NOM

#ifndef NOM

Créer un module

- Fournir un fichier d'entête .h pour utiliser les fonctionnalités du module.
- Implémenter les fonctionnalités dans un fichier .c du même nom.

Créer un module : fichier d'entête .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
```

Créer un module : implémentation .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);
}

```

Makefiles

Makefile

executable :

```
gcc -o executable *.c
```

make

Makefiles

Makefile

```
executable :
```

```
    gcc -o executable *.c
```

make

Makefiles

Un Makefile : liste de cibles, dépendances associées et commandes à exécuter. Chaque commande est précédée d'une **tabulation**.

cible: dépendances

 commande

 ...

 commande

Exemple de Makefile pour projet C

```
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
```

Symboles dans le Makefile

Symbol	Correspondance
\$@	Cible
\$^	Toutes les dépendances
\$<	Première dépendance
%	Règle générique

Exemple de Makefile pour projet C avec symboles

```
executable : main.o module.o
```

```
    gcc -o $@ $^
```

```
main.o : main.c module.h
```

```
    gcc -c $<
```

```
module.o : module.c module.h
```

```
    gcc -c $<
```

Exemple de Makefile pour projet C avec symboles et règle générique

```
executable : main.o module.o  
gcc -o $@ $^
```

```
main.o : main.c module.h  
module.o : module.c module.h
```

```
%.o : %.c  
gcc -c $<
```

Variables dans un Makefile

- définition VARIABLE=VALEUR
- appel \$(VARIABLE)

Pertinentes dans notre cas :

- le compilateur : CC=gcc (ou par exemple CC=clang).
- les drapeaux de compilation et optimisations dans CFLAGS (optimisations : -O1, -O2 et éventuellement -O3).
- les bibliothèques dans CLIBS.
- le nom de l'exécutable.

Makefile pertinent

```
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 $<
```

Makefile pertinent avec dossiers

```
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)
```

Makefile magique

```
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 $(SRC)%.*.c,$(OBJ)%.*.o,$(FILEC))

$(EXE) : $(FILEO)
        $(CC) $(CFLAGS) -o $@ $^ $(CLIBS)

$(OBJ)main.o : $(SRC)main.c $(FILEH)
        $(CC) $(CFLAGS) -o $@ -c $<

$(OBJ)%.*.o : $(SRC)%.*.c $(INCL)%.*.h
        $(CC) $(CFLAGS) -o $@ -c $<

clean :
        rm -rf $(OBJ)*.o
        rm -rf $(EXE)
```

Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur la programmation modulaire (section 13) du support de cours.
 - Si les exercices de la section 13 sont terminés :
 - Avancer sur les sections 14 et 15.
 - Si cours terminé : Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe



Programmation langage C

Section 14 : Opérations bit-à-bit

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Introduction

La machine travaille sur une représentation binaire des données.
Comment profiter de cette représentation binaire dans notre code ?

Opérateurs bit-à-bit

Il existe des opérateurs permettant de travailler en considérant la représentation binaire de nos entiers.

Décalage à gauche

```
/* décalage binaire à gauche */
nombre << nombreDeBitsADecaler
```

Équivalent à multiplier par une puissance de 2.

12	0	0	0	0	1	1	0	0
12 << 2 = 48	0	0	1	1	0	0	0	0

Décalage à gauche \star^1

Construction des puissances de 2 directement (sans fonction).

```
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);  
}
```

Décalage à gauche : dépassement de capacité \star^2

```
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);
```

Décalage à droite

```
/* décalage binaire à droite */  
nombre >> nombreDeBitsADecaler
```

Équivalent à une division entière par une puissance de 2.

24	0	0	0	1	1	0	0	0
24 >> 3 = 3	0	0	0	0	0	0	1	1

Négation bit-à-bit

/ change les valeurs de vérité de chaque bit */*
~nombre

7	0	0	0	0	0	1	1	1
~7	1	1	1	1	1	0	0	0

Négation bit-à-bit : nombres négatifs \star^3

Utilisé pour représentation d'un nombre négatif (complément à deux) :

```
int i = 42;  
printf("%d + %d = %d\n", i, ~i + 1, i + ~i + 1);
```

Négation bit-à-bit et décalage

Nombre avec tous les bits à 1 sauf celui voulu :

```
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);  
}
```

Négation bit-à-bit : limite de capacité \star^4

```
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);
```

Et bit-à-bit

3	0	0	1	1
6	0	1	1	0
2	0	0	1	0

Et bit-à-bit : exemple pratique - récupérer options

```
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);
}
```

Ou inclusif bit-à-bit

3	0	0	1	1
6	0	1	1	0
7	0	1	1	1

Ou inclusif bit-à-bit : exemple pratique - assembler options

★⁵

```
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");
```

Ou exclusif bit-à-bit

3	0	0	1	1
6	0	1	1	0
5	0	1	0	1

Ou exclusif bit-à-bit : exemple pratique - assembler options non partagées

```
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");
```

Ou exclusif bit-à-bit : échanger entier \star^6

```
int a = 42, b = 1337;
```

```
a ^= b; /* a ^ b */
```

```
b ^= a; /* b ^ a ^ b = a */
```

```
a ^= b; /* a ^ b ^ a = b */
```

Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur les opérations bit-à-bit (section 14) du support de cours.
- Si les exercices de la section 14 sont terminés :
 - Avancer sur la section 15.
 - Si cours terminé : Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe



(LAUGHING)

**Looking at
programming
memes**



(CRYING)

**Actually
coding**

Programmation langage C

Section 15 : Généricité

Présentation de **Kevin TRANCHO**

dispensé en classe de seconde année

à l'**ESGI** Paris
(Année scolaire 2022 - 2023)



école supérieure de
génie informatique

Introduction

C isn't that hard:

```
void (*(*f[])())());
```

Defines *f* as an array of unspecified size of pointers to functions that return pointers to functions with a return type of *void*.

Introduction

Comment factoriser du code d'un langage fortement typé comme le C avec des mécaniques plus génériques ?

Multitude 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));\n\nint 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; }
```

Multitude de fonctions : gestion disjonctive ? ★¹

```
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);
}
```

Multitude de fonctions : gestion disjonctive ?

Peu maintenable :

- Redondance et duplication pour chaque utilisation.
- Nécessité de rechercher chaque portion du code qui en fait usage (avec risque d'oubli).
- Dans l'idéal, un tableau ? Comment référencer ces fonctions ?

Multitude de fonctions : gestion disjonctive ?

Peu maintenable :

- Redondance et duplication pour chaque utilisation.
- Nécessité de rechercher chaque portion du code qui en fait usage (avec risque d'oublis).
- Dans l'idéal, un tableau ? Comment référencer ces fonctions ?

Multitude de fonctions : gestion disjonctive ?

Peu maintenable :

- Redondance et duplication pour chaque utilisation.
- Nécessité de rechercher chaque portion du code qui en fait usage (avec risque d'oublis).
- Dans l'idéal, un tableau ? Comment référencer ces fonctions ?

Pointeur de fonction : type référençant une fonction

```
typeRetour (* nom)(typeArguments);
```

Pointeur de fonction : type référençant une fonction

```
int (* operateurBinaire)(int, int) = &addition;  
operateurBinaire = &soustraction;
```

Pointeur de fonction : type référençant une fonction

```
/* autorisé par le compilateur : */
int (* operateurBinaire)(int, int) = addition;
operateurBinaire = soustraction;
```

Pointeur de fonction : appel

```
(*operateurBinaire)(1, 1); /* valide */  
operateurBinaire(1, 1);      /* autorisée */
```

Construire association fonction - nom - symbole

```
#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;
        }
    }
}
```

Utilisation association fonction - nom - symbole

```
/* Ajoutez vos opérations ici : */
OperationBinaire operations[] = {
    LISTOP(addition, +),
    LISTOP(soustraction, -),
    LISTOP(multiplication, *),
    LISTOP(division, /),
    LISTOP(modulo, %),
    LISTOP(decalageGauche, <<),
    LISTOP(decalageDroite, >>),
    {NULL}
};
```

Utilisation association fonction - nom - symbole \star^2

```
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));  
}
```

Pointeurs de fonctions : constructions plus complexes

Le langage C est riche de concepts : tableaux, pointeurs, fonctions.
Comment les utiliser avec des pointeurs de fonctions ?

Pointeurs de fonctions : constructions plus complexes

```
/* type : */  
typeRetour (* nom)(typeArguments);
```

```
/* tableau statique : */  
typeRetour (* nom[TAILLE])(typeArguments);
```

```
/* pointeur : */  
typeRetour (** nom)(typeArguments);
```

```
/* fonction (type de retour) : */  
typeRetour (* nom(ArgumentsFonction))(typeArguments);
```

Pointeurs de fonctions : tableau statique

```
int (* tableau[])(int, int) = {  
    addition,  
    soustraction,  
    NULL  
};  
  
tableau[1](2, 1);
```

Pointeurs de fonctions : allocation dynamique

```
int (** pointeur)(int, int) = NULL;  
if((pointeur = (int (**)(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);
```

Pointeurs de fonctions : fonction

```
int (* selectOperation(const char * symbole))(int,  
→ int) {  
    if(strcmp(symbole, "+") == 0)  
        return addition;  
    else if(strcmp(symbole, "-") == 0)  
        return soustraction;  
    return NULL;  
}
```

Pointeurs de fonctions : construction complexe

```
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);
}
```

Pointeurs de fonctions : construction complexe

```
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))malloc(sizeof(mapToIntToInt (*)(int)));
    *ppfc = fc;
    printf("%d\n", (*ppfc)(0)(4));
    exit(EXIT_SUCCESS);
}
```

Pointeurs de fonctions : construction complexe

```
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
        *)malloc(sizeof(selectMITIFromInt));
    *ppfc = fc;
    printf("%d\n", (*ppfc)(0)(4));
    exit(EXIT_SUCCESS);
}
```

Pointeur générique

- Typage du C problématique : dupliquer le code pour l'utiliser avec différents type ?
- Une adresse est de même taille peu importe le type pointé.
- Pointeur générique : void *

Pointeur générique

- Typage du C problématique : dupliquer le code pour l'utiliser avec différents type ?
- Une adresse est de même taille peu importe le type pointé.
- Pointeur générique : void *

Pointeur générique

- Typage du C problématique : dupliquer le code pour l'utiliser avec différents type ?
- Une adresse est de même taille peu importe le type pointé.
- Pointeur générique : void *

Pointeur générique : trier des éléments avec qsort

```
# 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 *));
```

Pointeur générique : trier des éléments avec qsort

```
int intcmp(const void * firstAddress, const void *
→ secondAddress) {
    int first = *((int *)firstAddress);
    int second = *((int *)secondAddress);
    return first - second; /* first < second <=> first -
→ second < 0 */
}

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);
}
```

Pointeur générique : trier des éléments avec qsort

```
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);
}
```

Pointeur générique : structure de données générique ?

- Connaître la taille d'un élément.
- Fournir fonctionnalités particulières sur un élément (affichage, allocation, libération et autres).
- L'arithmétique des pointeurs pas automatique. Nécessite taille élément. Manipulation possible avec `unsigned char *` (1 octet d'offset).

Pointeur générique : structure de données générique ?

- Connaître la taille d'un élément.
- Fournir fonctionnalités particulières sur un élément (affichage, allocation, libération et autres).
- L'arithmétique des pointeurs pas automatique. Nécessite taille élément. Manipulation possible avec `unsigned char *` (1 octet d'offset).

Pointeur générique : structure de données générique ?

- Connaître la taille d'un élément.
- Fournir fonctionnalités particulières sur un élément (affichage, allocation, libération et autres).
- L'arithmétique des pointeurs pas automatique. Nécessite taille élément. Manipulation possible avec `unsigned char *` (1 octet d'offset).

Liste générique : structure

```
typedef struct Liste Liste;
struct Liste {
    void * data;
    long taille;
    long capacite;
    size_t elementTaille;
    void (* elementFree)(void *);
    void (* elementPrint)(void *);
};
```

Liste générique : allocation

```
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 */
    }
```

Liste générique : libération

```
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;
}
```

Liste générique : affichage

```
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");  
}
```

Liste générique : ajout

```
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;
}
```

Liste générique : définitions utilisateur

```
void intPrint(int * value) {  
    printf("%d", *value);  
}
```

Liste générique : utilisation ★³

```
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);
}
```

Fonction à arguments variadiques

```
#include <stdarg.h>

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 */
}
```

Fonction à arguments variadiques : moyenne

```
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;
}
```

Fonction à arguments variadiques : moyenne \star^4

```
int main() {  
    printf("%g\n", moyenne(3, 10., 11., 13.));  
    exit(EXIT_SUCCESS);  
}
```

Questions

Avez-vous des questions ?

Exercices

- Travailler sur les exercices sur la générnicité (section 15) du support de cours.
 - Si les exercices de la section 15 sont terminés :
 - Avancer sur le projet.
 - Si projet terminé avec certitude de 21 / 20 : le pousser plus loin.

Annexe

