UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

2019-2020

télécom saint-étienne
école d'ingénieurs nouvelles technologies

CiTiSE1 et S1

IUT SAINT-ETIENNE

Info1

Info1: Fascicule d'exercices

Florent Bernard, Fabien Momey

Année 2019-2020

Table des matières

1 Algorithmie : premières notions **6**
1.1 [Ch] Écluse [1] . . . 6
1.2 [TD] Rigueur et précision (*) . . . 6
1.3 [Corr] Écluse . . . 7

2 La décomposition simple **8**
2.1 [Ch] Algorithmes erronés . . . 8
2.2 [Ch] Opérateurs . . . 9
2.3 [TD] Un classique (*) . . . 9
2.4 [TD] Superficie d'un terrain . . . 10
2.5 [TD] Heures/minutes/secondes ⟶ secondes . . . 10
2.6 [TD] Moyenne arithmétique . . . 10
2.7 [TD] Conversion Francs ⟶ Euros . . . 10
2.8 [TD] TVA (*) . . . 10
2.9 [TD] Circuit RLC (*) . . . 10
2.10 [TD] Interpolation linéaire (*) . . . 10
2.11 [TD] Secondes⟶ Heures/minutes/secondes (**) . . . 11
2.12 [TD] Retour de vacances (*) . . . 11
2.13 [Corr] Algorithmes erronés . . . 12
2.14 [Corr] Opérateurs . . . 13

3 Les fonctions **14**
3.1 [Ch] Erreurs classiques mais graves . . . 14
3.2 [Ch] Conversion Kms⟶Miles . . . 15
3.3 [Ch] Découpage fonctionnel donné . . . 15
3.4 [TD] Fonctions de traitement (*) . . . 16
3.5 [TD] Volume d'une sphère (*) . . . 16
3.6 [TD] Réutilisation de fonctions de l'exercice 3.3 (*) . . . 16
3.7 [TD] Découpage fonctionnel (**) . . . 16
3.8 [Corr] Erreurs classiques mais graves . . . 18
3.9 [Corr] Conversion Kms⟶Miles . . . 20
3.10 [Corr] Découpage fonctionnel donné . . . 21

1. Exercice proposé par Arnaud Giersch, enseignant à l'IUT Informatique de Belfort-Montbéliard.

4 Codage en langage C **26**
4.1 [Ch] Prise en main de Visual Studio . . . 27
4.2 [TD] Prise en main de Code::Blocks . . . 27
4.3 [TD] Débogage d'un code . . . 28
4.4 [TD] Traduction d'algorithmes . . . 29
4.5 [TD] Sans gravité . . . 29
4.6 [TD] Opérateurs mathématiques complexes avec les fonctions déclarées dans `math.h` . . . 30

5 Les fonctions en C **31**
5.1 [TD] A partir du cours . . . 31
5.2 [TD] Traduction de fonctions simples (*) . . . 31
5.3 [TD] Découpage fonctionnel . . . 32
5.4 [TD] Durée d'ensoleillement (*) . . . 32
5.5 [TD] Heures-minutes-secondes (**) . . . 32
5.6 [TD] Courbure de la terre et horizon visible (***) . . . 32

6 L'analyse par cas **34**
6.1 [Ch] Sur les conditions logiques . . . 34
6.2 [Ch] Quelques applications directes . . . 34
6.3 [TD] Interpolation linéaire . . . 35
6.4 [TD] Taille génétique . . . 35
6.5 [TD] Conditions multiples (*) . . . 35
6.6 [TD] Imbrication des Si (**) . . . 36
6.7 [Ch] Le Selon . . . 36
6.8 [TD] Pour aller plus loin (**) . . . 37
6.9 [Corr] Sur les conditions logiques . . . 37
6.10 [Corr] Quelques applications directes . . . 38
6.11 [Corr] Le Selon . . . 40

7 L'analyse par cas en C **45**
7.1 [TD] Traduction du Si en C avec des fonctions (*) . . . 45
7.2 [TD] Traduction du Selon en C avec des fonctions (**) . . . 45
7.3 [TD] Algo + C (**) . . . 46

8 Les structures itératives **47**
8.1 [Ch] Boucles élémentaires . . . 47
8.2 [TD] Séance de tirs au but (**) . . . 47
8.3 [TD] Racine carrée (**) . . . 48
8.4 [TD] Suite de Fibonacci (**) . . . 48
8.5 [TD] PGCD (**) . . . 48
8.6 [TD] Tables de multiplication (**) . . . 48
8.7 [TD] Kakuro : nombres fléchés (***) . . . 49
8.8 [Corr] Boucles élémentaires . . . 51

9 Les structures itératives en C **54**
9.1 [TD] PGCD (**) . . . 54
9.2 [TD] Séance de tirs au but (**) . . . 54
9.3 [TD] Racine carrée (**) . . . 55
9.4 [TD] Suite de Fibonacci (**) . . . 55
9.5 [TD] Tables de multiplication (**) . . . 55
9.6 [TD] Suite de Syracuse (***) . . . 56
9.7 [TD] Dessins : motifs (**) . . . 56
9.8 [TD] Dessins : ensemble de points (**) . . . 58
9.9 [TD] Défi Kangourou (****) . . . 58

10 Les tableaux **60**
10.1 [Ch] A partir d'algorithmes donnés . . . 60
10.2 [Ch] Saisie d'un nombre inconnu de valeurs dans un tableau . . . 61
10.3 [TD] Utilisation directe de tableaux . . . 61
10.4 [TD] Avec des Si (*) . . . 61
10.5 [TD] Manipulation de tableaux et boucles (*) . . . 62
10.6 [TD] Tableaux à deux dimensions et boucles (**) . . . 62
10.7 [TD] Tableaux à deux dimensions (**) . . . 63
10.8 [TD] Interpolation linéaire dans un tableau (**) . . . 63
10.9 [TD] Chaînes de caractères et tableaux (*) . . . 64
10.10 [TD] Chaînes de caractères (*) . . . 64
10.11 [TD] Conversions nombre arbitrairement grand en tableau d'entiers (****) . . . 64
10.12 [TD] Conversions nombre sous forme de tableau d'entiers en chaîne de caractères (****) . . . 65
10.13 [Corr] A partir d'algorithmes donnés . . . 65
10.14 [Corr] Saisie d'un nombre inconnu de valeurs dans un tableau . . . 67

11 Les tableaux en C **70**
11.1 [TD] Fonctions de manipulation de tableaux . . . 70
11.2 [TD] Tableaux et chaînes de caractères . . . 70
11.3 [TD] Traduction des algorithmes du chapitre 10 . . . 70

12 Le type composé **71**
12.1 [Ch] A partir d'un algorithme donné . . . 71
12.2 [Ch] Points . . . 71
12.3 [Ch] Menus . . . 72
12.4 [TD] Points et vecteurs (*) . . . 73
12.5 [TD] Complexes (*) . . . 73
12.6 [TD] type composé, tableaux et boucles (***) . . . 73
12.7 [TD] Synthèse (d'après sujet examen 2014-2015) . . . 73
12.8 [TD] Les 12 billes (****) . . . 74
12.9 [Corr] A partir d'un algorithme donné . . . 75
12.10 [Corr] Points . . . 77
12.11 [Corr] Menus . . . 78

13 Le type composé en C **84**
13.1 [C] Ensemble de fonctions sur les complexes 84
13.2 Application 84
13.3 Traduction des exercices du chapitre 12 85

14 Les pointeurs **86**

Introduction

Ce fascicule contient des exercices donnés pendant le cours d'Info1 du 1er semestre aux étudiants du département GEII de l'IUT de Saint-Etienne (CiTiSE1 ou S1).
Les exercices sont regroupés par chapitre et suivent le même plan que le support de cours distribué au format `.pdf` en début d'année aux CiTiSE.
Différents sigles sont utilisés dans ce document :

- [Ch] Exercices à chercher en autonomie par l'étudiant lors de l'étude du chapitre de cours correspondant.
- [Corr] Correction mise à disposition dans ce fascicule des exercices à chercher.
- [TD] Exercices à chercher en séance de TD et à terminer chez soi. Aucune correction tapée ne sera proposée pour ces exercices mais des questions pourront être posées par les étudiants d'une séance de TD sur l'autre ou en séance de CM.

Un nombre d'* (de 0 à 4) a été attribué aux exercices de [TD] :

- (aucune) application directe du cours. A savoir faire et refaire impérativement.
- (*) une très légère difficulté à surmonter ou appel à des connaissances générales sensées être maîtrisées et non revues dans le cours. A savoir faire et refaire impérativement.
- (**) une légère difficulté algorithmique nécessitant une phase de reflexion amont sur papier et/ou des tests. A refaire et à maitriser après la correction.
- (***) une difficulté réelle algorithmique nécessitant une phase approfondie de reflexion amont sur papier et des tests. A refaire après la correction pour progresser.
- (****) exercice demandant une reflexion très approfondie et de nombreux tests d'essais/erreurs/correction pour parvenir au résultat. A refaire et à maîtriser pour ceux qui se destinent à un parcours plutôt orienté informatique.

Pour les exercices `[Ch]` aucune `*` n'est attribuée puisque ce sont des applications directes du cours dont la correction `[Corr]` figure dans ce fascicule.

Chapitre 1

Algorithmie : premières notions

1.1 [Ch] Écluse [1]

Les portes d'une écluse sont munies de vannes que l'on peut ouvrir ou fermer et qui permettent de laisser passer l'eau afin de mettre l'écluse à niveau (Fig. 1.1). Les portes sont également équipées de feux pouvant être vert ou rouge, pour autoriser ou non le passage du bateau.

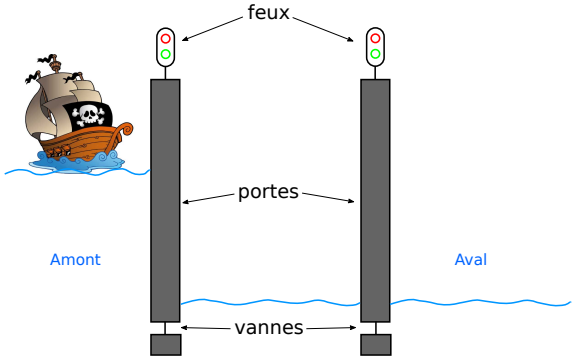


FIGURE 1.1 – Schéma d'une écluse.

On considérera en situation initiale que le bateau descend la rivière et que lorsqu'il se présente devant l'écluse, celle-ci est au niveau bas, que les portes et vannes sont fermées et que les feux sont au rouge.

— Indiquez les données d'entrée et la suite d'actions permettant de modéliser le franchissement d'une écluse par un bateau.

1.2 [TD] Rigueur et précision (*)

Proposer une liste de matériel nécessaire et la suite d'actions à effectuer dans un ordre donné en utilisant ce matériel pour permettre à une personne quelconque de dessiner un carré.

1. Exercice proposé par Arnaud Giersch, enseignant à l'IUT Informatique de Belfort-Montbéliard.

1.3 [Corr] Écluse

Indiquez les données d'entrée et la suite d'actions permettant de modéliser le franchissement d'une écluse par un bateau.

- **Énoncé** :
  - ▷ Faire franchir une écluse à un bateau.
- **Données et accessoires** :
  - ▷ Un bateau se présente devant l'écluse en amont de la rivière.
  - ▷ L'écluse est au niveau bas.
- **Actions détaillées** :
  - ▷ Ouvrir vanne amont.
  - ▷ Lorsque le niveau de l'écluse est au niveau de la rivière en amont (niveau haut), fermer vanne amont.
  - ▷ Ouvrir porte amont.
  - ▷ Feu amont au vert.
  - ▷ Faire entrer le bateau dans l'écluse jusqu'à la porte aval.
  - ▷ Feu amont au rouge.
  - ▷ Fermer porte amont.
  - ▷ Ouvrir vanne aval.
  - ▷ Lorsque le niveau de l'écluse est au niveau de la rivière en aval (niveau bas), fermer vanne gauche.
  - ▷ Ouvrir porte aval.
  - ▷ Feu aval au vert.
  - ▷ Faire passer le bateau en aval de la rivière.
  - ▷ Feu aval au rouge.
  - ▷ Fermer porte aval.

Chapitre 2

La décomposition simple

2.1 [Ch] Algorithmes erronés

On donne les algorithmes suivants qui comportent des erreurs classiques vues dans des copies d'étudiants.
Identifiez les erreurs (afin de ne plus les refaire) et corrigez les.

Algo1

Lexique :
taux : 1 constante réelle
frcs : 1 entier

Algorithme :
taux⟵ 6.55957
frcs⟵ taux*55

Algo2

Lexique :
EURTOFR : la constante réelle :=6.55957
frcs, eur : 2 réels

Algorithme :
Début
Ecrire("Entrez votre nombre d'euros")
frcs⟵ EURTOFR*eur
Ecrire(eur,"euros = ",frcs, "francs")

Algo3

Lexique :
PI : la constante réelle :=3.14159
p : 1 entier

Algorithme :
Début
Ecrire("Entrez votre rayon (cm)")
Lire(r)
p⟵ 2*r*PI

Ecrire(“Le périmètre du cercle de rayon”, r, “cm est de ”,p, “cm”)
Fin

voir la correction : 2.13

2.2 [Ch] Opérateurs

Algo1

Lexique :
a,b :2 entiers

Algorithme :
Début
a⟵ 5
b⟵ a-6
a⟵ a+1
b⟵ b*4
a⟵ -2*b-4
a⟵ a/3
Fin

1. Donner après chaque instruction les valeurs des variables a et b.
2. La dernière instruction a⟵a/3 est-elle autorisée ? Pourquoi ?
3. Proposer une instruction permettant d'avoir le quotient de a par 3.

Opérateurs `div` et `reste`

Donner le résultat attendu des opérations suivantes :

1. 1725 `div` 34 et 1725 `reste` 34
2. 4397 `div` 60 et 4397 `reste` 60
3. 47 `div` 50 et 47 `reste` 50

Quel résultat donnerait l'opération 1725/34 ?

voir la correction : 2.14

2.3 [TD] Un classique (*)

Proposer un algorithme permettant de demander à un utilisateur de saisir deux nombres, de les mémoriser dans deux variables `a` et `b` puis d'échanger le contenu de deux variables. Vous afficherez un message indiquant le contenu de `a` et `b` avant échange et un autre message indiquant le contenu de `a` et `b` après échange. [1] .

	avant échange		après échange	
Exemple :	3	5	5	3
	a	b	a	b

1. Il est possible de réaliser cet exercice en utilisant uniquement les deux variables a et b au prix de calculs arithmétiques supplémentaires (proposition de Yann Moulaire, CiTiSE1 2015-2016). Toutefois le coût en terme de calculs supplémentaires se discute et il est préférable d'utiliser une variable de plus. La solution n'en reste pas moins astucieuse ! Avez-vous trouvé comment a fait Yann ?

2.4 [TD] Superficie d'un terrain

Proposer un algorithme qui permet à un utilisateur de saisir la longueur et la largeur d'un terrain rectangulaire en mètres, puis de calculer et afficher sa superficie en hectares. Rappel : 1 hectare (ha) correspond à 1 hectomètre carré (hm^2). 1 hectomètre carré correspond à 10^4 m^2.

2.5 [TD] Heures/minutes/secondes $\longrightarrow$ secondes

Ecrire un algorithme permettant à un utilisateur de saisir un temps sous la forme `hh:mm:ss` et qui affiche le nombre total de secondes.

2.6 [TD] Moyenne arithmétique

Ecrire un algorithme permettant à un utilisateur de saisir 3 notes et d'afficher la moyenne arithmétique de ces trois notes.

2.7 [TD] Conversion Francs $\longrightarrow$ Euros

Ecrire un algorithme permettant à un utilisateur de saisir une somme en francs et d'afficher la somme convertie en euros.

2.8 [TD] TVA (*)

Ecrire un algorithme qui propose à l'utilisateur d'entrer le prix hors taxe unitaire d'un article, la quantité d'article commandée, un taux de TVA en pourcentage et qui calcule et affiche le prix TTC total de la commande de ces articles.

2.9 [TD] Circuit RLC (*)

Proposer un algorithme qui permet de calculer l'impédance équivalente d'un circuit comportant en série une résistance, une inductance et un condensateur à une fréquence donnée. On demandera la saisie des diverses grandeurs physiques par l'utilisateur.
Astuce : un nombre complexe peut être séparé en une partie réelle et une partie imaginaire.

2.10 [TD] Interpolation linéaire (*)

Soient 2 points A et B définis dans un repère cartésien $\{O, x, y\}$ par leurs coordonnées respectives $\{x_A, y_A\}$ et $\{x_B, y_B\}$. L'**interpolation linéaire** au point C d'abscisse $x_C \in [x_A, x_B]$ consiste à déterminer son ordonnée y_C telle que les points A, B et C soient alignés (ils appartiennent à la même droite). La figure 2.1 illustre la situation sus-décrite.

Voici la formule type de l'interpolation linéaire au point C :

$$y_C = \alpha \; y_B + (1 - \alpha) \; y_A \; ,$$

avec $\alpha = \frac{x_C - x_A}{x_B - x_A}$.

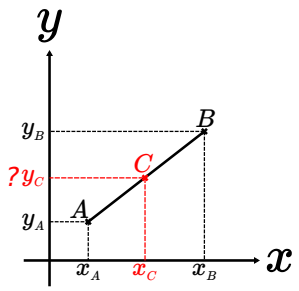


FIGURE 2.1 – Interpolation linéaire au point C entre les 2 points A et B

Proposer un algorithme qui permet de saisir les coordonnées des points A et B, puis de réaliser leur interpolation linéaire au point C d'abscisse x_C (qui sera demandée en saisie également).
On suppose, que le point saisi a une abscisse qui est bien comprise entre x_A et x_B.

2.11 [TD] Secondes⟶ Heures/minutes/secondes (**)

Ecrire un algorithme permettant à un utilisateur de saisir un temps en secondes et d'afficher ce temps sous la forme `hh:mm:ss`

2.12 [TD] Retour de vacances (*)

Une famille part en vacances en voiture et initialise son ordinateur de bord en début de vacances. Au retour, l'ordinateur de bord de la voiture indique le nombre total de kilomètres effectués, la consommation moyenne en L/100kms, ainsi que la vitesse moyenne. Proposer un algorithme qui permet de saisir ces informations, ainsi que le prix moyen du litre d'essence afin de donner (afficher), à la famille des informations sur :

— Le temps passé dans la voiture
— Le coût en carburant des vacances

Vous testerez votre programme avec les valeurs suivantes

2.13 [Corr] Algorithmes erronés

On donne les algorithmes suivants qui comportent des erreurs classiques vues dans des copies d'étudiants.
Identifiez les erreurs (afin de ne plus les refaire) et corrigez les.

Algo1

Lexique :

~~taux~~ TAUX : ~~1~~ la constante réelle := 6.55957 {Bien penser à l'initialisation, les constantes se notent en MAJUSCULE}

frcs : 1 ~~entier~~ réel {Ce n'est pas une instruction fausse en tant que telle. Mais le résultat du calcul effectué dans l'algorithme est un réel donc doit être stocké dans une variable de même type.}

Algorithme :

Début {A ne pas oublier !}

~~taux~~ ⟵ ~~6.55957~~ {grosse erreur : une constante n'est JAMAIS l'objet d'une affectation}

frcs⟵ ~~taux~~ TAUX*55

Fin {A ne pas oublier !}

Même si ce ne sont pas des erreurs à proprement parler, il manque les instructions de communication avec l'utilisateur de l'algorithme `Lire` et `Ecrire` pour rendre l'algorithme utilisable.

Algo2

Lexique :

EURTOFR : la constante réelle :=6.55957

frcs, eur : 2 réels

Algorithme :

Début

Ecrire("Entrez votre nombre d'euros")

Lire(eur) {On récupère la saisie de l'utilisateur et on la mémorise dans la variable eur}

frcs⟵ EURTOFR*eur

Ecrire(eur,"euros = ",frcs, "francs")

Fin {A ne pas oublier !}

Algo3

Lexique :

PI : la constante réelle :=3.14159

p : 1 ~~entier~~ réel

r : 1 réel {Ne pas oublier de déclarer toutes les variables utilisées dans l'algorithme}

Algorithme :

Début

Ecrire("Entrez votre rayon (cm)")

Lire(r)

p⟵ 2*r*PI
Ecrire("Le périmètre du cercle de rayon", r, "cm est de ",p, "cm")
Fin

Retour à l'enoncé : 2.1

2.14 [Corr] Opérateurs

Algo1

Lexique :
a,b : 2 entiers

Algorithme :
Début

a⟵ 5	{a contient 5 et b ne contient aucune valeur}
b⟵ a-6	{a contient 5 et b contient -1}
a⟵ a+1	{a contient 6 et b contient -1}
b⟵ b*4	{a contient 6 et b contient -4}
a⟵ -2*b-4	{a contient 4 et b contient -4}
~~a⟵ a/3~~ a⟵a `div` 3	{a contient 1 et b contient -4}

Fin

1. voir les commentaires après chaque instruction
2. La dernière instruction a⟵a/3 n'est pas autorisée : a/3 retourne un réel et a est un entier. On ne peut pas affecter un réel dans un entier selon le principe de respect du type de données lors d'une affectation.
3. si on veut le quotient de de a par 3, on peut utiliser l'opérateur `div`. Voir l'instruction correspondante dans l'algorithme.

Opérateurs `div` et `reste`

Donner le résultat attendu des opérations suivantes :

1. 1725 `div` 34 = 50 et 1725 `reste` 34 = 25 (1725=50×34+25)
2. 4397 `div` 60 = 73 et 4397 `reste` 60 = 17 (4397=73×60+17)
3. 47 `div` 50 = 0 et 47 `reste` 50 = 47 (47=0×50+47)

1725/34 est une division réelle, le résultat est donc un réel qui est à peu près égal à 50.7352941

Retour à l'enoncé : 2.2

Chapitre 3

Les fonctions

Dans ces exercices et plus globalement, dès que vous utiliserez des fonctions, on vous demande de préciser avant toute chose et pour chaque fonction utilisée :

— son rôle ;
— ses paramètres d'entrée (en particulier leurs types) ;
— son type de retour.

Ceci sera systématiquement fait dans le cas d'un découpage fonctionnel, mais doit aussi être fait dans le cas de fonctions isolées.

3.1 [Ch] Erreurs classiques mais graves

Voici un Lexique principal ainsi qu'un algorithme principal (les deux sont à complèter) permettant d'utiliser une fonction de conversion d'un temps donné en heures/minutes/secondes en un temps en secondes.

Lexique :{principal}
Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier
{Lexique principal à compléter}

Algorithme :{principal}
Début
{A compléter}
Fin

Voici la définition proposée par un étudiant.

Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier
  Lexique :{Local à Conv_h_m_s}
    hh, mm, ss : 3 entier
    sec : 1 entier
  Algorithme :{Local à Conv_h_m_s}
  Début
    Ecrire("Entrer le nombre d'heures")
    Lire(hh)
    Ecrire("Entrer le nombre de minutes")
    Lire(mm)

```
  Ecrire("Entrer le nombre de secondes")
  Lire(ss)
  sec⟵hh*3600+mm*60+ss
  Ecrire("Le nombre de secondes est :",sec)
Fin
```

Identifier et corriger les erreurs, puis compléter le lexique principal et l'algorithme principal permettant d'utiliser cette fonction.

Voir la correction : 3.8

3.2 [Ch] Conversion Kms⟶Miles

1. Dans un Lexique principal, déclarer la fonction `Km2miles` dont le rôle est de convertir des kilomètres en miles ((R)appel : 1 mile = 1.609 km).
2. Compléter le lexique principal puis proposer un algorithme principal dans lequel vous utiliserez cette fonction de conversion.
3. Proposer enfin un algorithme définissant cette fonction.

Voir la correction : 3.9

3.3 [Ch] Découpage fonctionnel donné

1. Proposer un découpage fonctionnel pour le problème suivant : "demander à un utilisateur de saisir le rayon d'un disque et lui calculer puis afficher le périmètre et la surface du disque"
   Une proposition de découpage a été faite en cours :
   (a) `Presentation`, rôle : expliquer ce que fait le programme, entrée : vide, sortie : vide
   (b) `Saisie_rayon`, rôle : demande à l'utilisateur d'entrer un rayon, entrée : vide, sortie : 1 réel (le rayon)
   (c) `Perimetre`, rôle : calcule le périmètre d'un cercle de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (le périmètre)
   (d) `Surface`, rôle : calcule la surface d'un disque de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (la surface)
   (e) `Affiche_resultats`, rôle : afficher les valeurs du périmètre et de la surface, entrée : 2 réels (périmètre et surface), sortie : vide
2. Proposer le lexique principal ainsi que l'algorithme principal permettant d'utiliser ces fonctions pour répondre au problème posé.
3. Proposer ensuite un algorithme pour chacune de ces fonctions.
4. Quelles sont les fonctions de traitement dans ce découpage ?

Voir la correction : 3.10

3.4 [TD] Fonctions de traitement (*)

1. Reprendre les algorithmes des exercices 2.4, 2.6, 2.7, 2.8 et proposer uniquement les algorithmes des fonctions de **traitement** dans chacun de ces exercices.
2. Proposer un Lexique principal et un algorithme principal qui vous permettront de tester ces fonctions.
3. Quel est l'intérêt d'utiliser des fonctions ?
4. Pourquoi on ne peut pas utiliser [1] de fonction pour résoudre l'exercice 2.11 ?

3.5 [TD] Volume d'une sphère (*)

1. Proposer un algorithme pour la fonction permettant de calculer le volume d'une sphère de rayon r.
2. Proposer un Lexique principal et un algorithme principal permettant de tester cette fonction.

3.6 [TD] Réutilisation de fonctions de l'exercice 3.3 (*)

1. Proposer un nouvel algorithme pour la fonction dont le rôle est de calculer le volume d'un cylindre. Cette nouvelle fonction **DEVRA** utiliser la fonction `Surface` de l'exercice 3.3
   Proposer un Lexique principal et un algorithme principal permettant d'utiliser cette fonction.
2. Un vélo dispose d'un compte-tours qui s'incrémente [2] de 1 à chaque fois que la roue avant du vélo fait un tour. Proposer l'algorithme d'une fonction dont le rôle est de donner la distance parcourue par le vélo connaissant le nombre de tours effectué par la roue. Cette nouvelle fonction **DEVRA** utiliser la fonction `Perimetre` de l'exercice 3.3
   Proposer un Lexique principal et un algorithme principal permettant d'utiliser cette fonction.

3.7 [TD] Découpage fonctionnel (**)

La durée d'insolation (en heures), c'est à dire le temps qui s'écoule entre le moment où le soleil se lève et celui où il se couche, est obtenue, pour une latitude donnée et un jour de l'année donné, par la formule simplifiée suivante :

$$D_i = \frac{24}{\pi} \arccos(-\tan(Lat) \times \tan(Dec))$$

où

— Lat désigne la latitude d'un lieu en radians

1. pour le moment !
2. incrémenter = augmenter sa valeur d'une quantité donnée

— Dec désigne la déclinaison du soleil en radians, c'est à dire l'angle que forme le vecteur "centre de la terre $\rightarrow$ Soleil" avec le plan équatorial de la terre, au cours de l'année.
Dec est donnée par la formule simplifiée suivante où j désigne un jour de l'année $j = 1$ pour le 1er janvier.

$$Dec = \arcsin(0.398 \times \sin(0.0171915 \times j - 1.3962634)))$$

Les fonctions arccos et arcsin donnent des résultats en radians directement.

Proposez un découpage fonctionnel simple dans lequel un utilisateur pourra saisir une latitude en degrés (45.4333° pour Saint-Etienne par exemple) et un jour de l'année 2018 (entre 1 et 365) (j=246 pour le 3 septembre 2018) et qui lui affiche la durée d'ensoleillement en heures pour le jour choisi.

3.8 [Corr] Erreurs classiques mais graves

De nombreuses erreurs sont présentes dans cet exercice. Il est très important de bien les repérer, les comprendre et les corriger pour ne plus les faire. La compréhension de la totalité de la correction de cet exercice est fondamentale pour pouvoir travailler correctement avec les fonctions pour tout le reste du semestre.
La première est qu'il manque, rôle (R), paramètres d'entrée (E) et type de retour/sortie (S) de la fonction.

Il peut paraître rébarbatif de préciser rôle, paramètres d'entrée, type de retour mais souvent, les étudiants bloqués sur un problème sont incapables de donner ces informations de base.
A l'inverse, si vous êtes capables de formuler clairement ces informations, vous avez effectué plus de la moitié du travail de compréhension.
En stage (au S4), les tuteurs industriels déplorent que les étudiants soient incapables de préciser clairement ces informations avant de se lancer dans le codage.

Commençons par la définition de cette fonction :

{R : Convertit un temps au format `hh:mm:ss` en secondes
E : 3 entiers correspondant respectivement au nombre d'heures (h), le nombre de minutes (m), le nombre de secondes (s)
S : 1 entier correspondant au nombre total de secondes}

Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier
Lexique :{Local à Conv_h_m_s}
~~hh, mm, ss : 3 entier~~ {C'est probablement la plus grosse et la plus grave erreur que vous puissiez faire.}
{il **FAUT** travailler avec les paramètres d'entrée de la fonction intitulés ici : h, m et s. La fonction les connaît et il n'est pas question de les redéclarer dans le lexique.}
sec : 1 entier {En revanche, il faut bien déclarer un entier qui va servir à mémoriser le nombre total de secondes qu'on aura calculé grâce aux paramètres d'entrée h, m et s.}
Algorithme :{Local à Conv_h_m_s}
Début
{Si vous avez compris l'intérêt des paramètres d'entrée et l'importance de ne pas les redéclarer, vous comprendrez facilement pourquoi on barre les 6 instructions suivantes.
Dans le cas contraire, on répète que la fonction travaille avec h,m et s qui lui sont communiqués par le programme principal.
Le programme principal a pour rôle d'interagir avec l'utilisateur pour lui demander ces informations (voir la suite de la correction). Il communique ensuite les paramètres à la fonction pour lui sous-traiter le travail.
On suppose donc que ces paramètres sont connus au moment où la fonction est appelée, il n'est donc pas nécessaire de demander à l'utilisateur d'entrer des valeurs dans la fonction ! !}
~~Ecrire("Entrer le nombre d'heures")~~
~~Lire(hh)~~
~~Ecrire("Entrer le nombre de minutes")~~
~~Lire(mm)~~
~~Ecrire("Entrer le nombre de secondes")~~
~~Lire(ss)~~

{L'instruction suivante correspond au calcul de conversion que doit effectuer la fonction. Cependant, elle doit travailler sur ses paramètres d'entrée h, m et s.}

sec⟵ ~~hh~~ h*3600+~~mm~~ m*60+~~ss~~ s

{Le rôle de cette fonction est d'effectuer un calcul de conversion, **un traitement** et en aucun cas un affichage. Elle doit donc **retourner** le résultat de son calcul au programme principal qui a appelé la fonction.
Ce sera ensuite au programme principal d'afficher le résultat à l'utilisateur}

~~Ecrire("Le nombre de secondes est :",sec)~~

Retourner sec

Fin

Lexique :{principal}

{R : Convertit un temps au format `hh:mm:ss` en secondes
E : 3 entiers correspondant respectivement au nombre d'heures (h), le nombre de minutes (m), le nombre de secondes (s)
S : 1 entier correspondant au nombre total de secondes}

Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier

{La fonction est déclarée correctement, on retrouve bien les 3 paramètres d'entrée qui sont de type entier et auxquels on a donné des noms (h,m,s). Ce sont ces paramètres qui **doivent** être utilisés pour calculer le nombre de secondes. On retrouve également le type retourné par la fonction (instruction Retourner) qui est bien un entier. Il est tout à fait normal que son nom n'apparaisse pas, seul son type est important.}

{C'est dans l'algorithme principal qu'on va demander à l'utilisateur de saisir son temps au format `hh:mm:ss`. C'est donc ici qu'on va déclarer les variables permettant de mémoriser ces informations}

hh,mm, ss : 3 entiers
secondes : 1 entier {pour mémoriser la valeur retournée par la fonction Conv_h_m_s}

Algorithme :{principal}

Début

{On commence par demander à l'utilisateur d'entrer des informations}

Ecrire("Entrez votre temps en saisissant les heures, puis les minutes puis les secondes et en validant après chaque saisie")

{On récupère les saisies de l'utilisateur dans les 3 variables du Lexique principal : hh, mm et ss.}

Lire(hh,mm,ss)

{Les informations étant saisies on peut passer au traitement qui est assuré par la fonction Conv_h_m_s}

secondes⟵Conv_h_m_s(hh,mm,ss) {Appel/utilisation de la fonction}

{Le résultat du calcul effectué et retourné par la fonction est stocké dans la variable secondes. Il ne reste plus qu'à afficher le résultat à l'utilisateur.}

Ecrire(hh," :",mm," :",ss,"=",secondes,"sec")

Fin

Si on prend en compte toutes ces remarques et qu'on rédige la solution complètement, on obtient :

Lexique :{principal}

{R : Convertit un temps au format `hh:mm:ss` en secondes

```
    E : 3 entiers correspondant respectivement au nombre d'heures (h), le nombre de
      minutes (m), le nombre de secondes (s)
    S : 1 entier correspondant au nombre total de secondes}
   Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier
   hh,mm, ss : 3 entiers
   secondes : 1 entier

Algorithme :{principal}
Début
  Ecrire("Entrez votre temps en saisissant les heures, puis les minutes puis les secondes
et en validant après chaque saisie")
  Lire(hh,mm,ss)
  secondes←—Conv_h_m_s(hh,mm,ss) {Appel/utilisation de la fonction}
  Ecrire(hh," :",mm," :",ss,"=",secondes,"sec")
Fin

{Définition de la fonction}
 Conv_h_m_s : la fonction (h : 1 entier, m : 1 entier, s : 1 entier)→ 1 entier
  Lexique :{Local à Conv_h_m_s}
   sec : 1 entier
  Algorithme :{Local à Conv_h_m_s}
  Début
   sec←—h*3600+m*60+s
   Retourner sec
  Fin
```

Retour à l'énoncé : 3.1

3.9 [Corr] Conversion Kms—→Miles

```
Lexique :{principal}
    {R : Convertit une distance donnée en kilomètres en la distance équivalente en miles
     E : 1 réel correspondant à la distance exprimée en kilomètres (km)
     S : 1 réel correspondant à la distance convertie en miles}
 Km2miles : la fonction (km : 1 réel)→ 1 réel
 kms, miles : 2 réels

Algorithme :{principal}
Début
  Ecrire("Entrez votre distance en kilomètres")
  Lire(kms)
  miles←—Km2miles(kms)
  Ecrire(kms,"kms = ",miles," miles")
Fin

{Définition de la fonction}
```

Km2miles : la fonction (km : 1 réel)→ 1 réel
  Lexique :{Local à Km2miles}
   mile : 1 réel
  Algorithme :{Local à Km2miles}
  Début
   mile←—km/1.609
   Retourner mile
  Fin

Retour à l'énoncé : 3.2

3.10 [Corr] Découpage fonctionnel donné

1. Proposer un découpage fonctionnel pour le problème suivant : "demander à un utilisateur de saisir le rayon d'un disque et lui calculer puis afficher le périmètre et la surface du disque"
   Une proposition de découpage a été faite en cours :
   (a) `Presentation`, rôle : expliquer ce que fait le programme, entrée : vide, sortie : vide
   (b) `Saisie_rayon`, rôle : demande à l'utilisateur d'entrer un rayon, entrée : vide, sortie : 1 réel (le rayon)
   (c) `Perimetre`, rôle : calcule le périmètre d'un cercle de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (le périmètre)
   (d) `Surface`, rôle : calcule la surface d'un disque de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (la surface)
   (e) `Affiche_resultats`, rôle : afficher les valeurs du périmètre et de la surface, entrée : 2 réels (périmètre et surface), sortie : vide
2. Lexique :{principal}
   {R : Expliquer ce que fait le programme
   E : vide
   S : vide}
   `Presentation` : la fonction (vide)→ vide

   {R : Demande à l'utilisateur d'entrer un rayon et retourne la valeur saisie
   E : vide
   S : 1 réel (le rayon)}
   `Saisie_rayon` : la fonction (vide)→ 1 réel

   {R : Calcule le périmètre d'un cercle de rayon r
   E : 1 réel (le rayon)
   S : 1 réel (le périmètre)}
   `Perimetre` : la fonction (r : 1 réel)→ 1 réel

{R : Calcule la surface d'un disque de rayon r
E : 1 réel (le rayon)
S : 1 réel (la surface)}
`Surface` : la fonction (r : 1 réel)→ 1 réel

{R : Afficher les valeurs du périmètre et de la surface
E : 2 réels (le périmètre et la surface)
S : vide}
`Affiche_resultats` : la fonction (p : 1 réel, s : 1 réel)→ vide
rayon, peri, surf : 3 réels

Algorithme :{principal}
Début
`Presentation()`
rayon⟵`Saisie_rayon`()
peri⟵`Perimetre`(rayon)
surf⟵`Surface`(rayon)
`Affiche_resultats`(peri,surf)
Fin

Passons aux définitions des fonctions :

`Presentation` : la fonction (vide)→ vide
Lexique :{Local à `Presentation`}
{vide}
Algorithme :{Local à `Presentation`}
Début
Ecrire("Ce programme vous permet de saisir le rayon d'un disque à partir duquel, le périmètre et la surface du disque sont calculés puis affichés")
Fin

{Il est très important de comprendre le prototype de la fonction suivante. Son rôle est d'interagir avec l'utilisateur pour lui demander d'entrer une information (ici le rayon). Il est donc tout à fait normal que cette fonction ne prenne aucun paramètre en entrée. En revanche l'information saisie par l'utilisateur doit être mémorisée par la fonction (à travers une variable locale) et retournée au programme principal. Le type de retour est donc un réel correspondant au rayon saisi par l'utilisateur.}
`Saisie_rayon` : la fonction (vide)→ 1 réel
Lexique :{Local à `Saisie_rayon`}
rayon : 1 réel
Algorithme :{Local à `Saisie_rayon`}
Début
Ecrire("Saisissez le rayon du disque en cm") {on indique à l'utilisateur ce qu'on attend de lui}
Lire(rayon) {On sauvegarde sa saisie dans la variable locale rayon (déclarée dans le lexique)}

Retourner rayon {On retourne cette valeur au programme principal (puisque rayon n'est connue **que** de la fonction `Saisie_rayon`}
Fin

{Pour les deux fonctions suivantes, ce sont des fonctions de traitement. On ne demande aucune information à l'utilisateur. Le rayon est un paramètre de la fonction qui lui sera communiqué par le programme principal.}
`Perimetre` : la fonction (r : 1 réel)→ 1 réel
Lexique :{Local à `Perimetre`}
p : 1 réel
PI : La constante réelle :=3.14159265
Algorithme :{Local à `Perimetre`}
Début
p←—2*PI*r
Retourner p
Fin

`Surface` : la fonction (r : 1 réel)→ 1 réel
Lexique :{Local à `Surface`}
s : 1 réel
PI : La constante réelle :=3.14159265
Algorithme :{Local à `Surface`}
Début
s←—PI*r*r
Retourner s
Fin

`Affiche_resultats` : la fonction (p : 1 réel, s : 1 réel)→ vide
Lexique :{Local à `Affiche_resultats`}
{vide}
Algorithme :{Local à `Affiche_resultats`}
Début
Ecrire("Le périmètre de votre disque est :", p, "cm")
Ecrire(" et sa surface est : ", s, "cm^2")
Fin

3. Les fonctions de traitement dans ce découpage sont `Perimetre` et `Surface`

Remarque 1 *Découpage fonctionnel* **simple**
La plupart du temps, on demandera un découpage fonctionnel simple. C'est à dire que tous les messages d'affichage ou de saisie pouvant être effectués par les fonctions de base `Ecrire()` *et* `Lire()`*, ne feront pas l'objet de fonctions particulières. De ce fait, la correction pourrait être allégée en proposant le découpage fonctionnel* **simple** *suivant et en précisant à quel niveau seront utilisées les fonctions* `Lire()` *et* `Ecrire()`

1. `Presentation`*, rôle : expliquer ce que fait le programme, entrée : vide, sortie : vide,* **assuré par la fonction** `Ecrire()`

2. `Saisie_rayon`, *rôle : demande à l'utilisateur d'entrer un rayon, entrée : vide, sortie : 1 réel (le rayon),* **assuré par la fonction** `Lire()`
3. `Perimetre`, *rôle : calcule le périmètre d'un cercle de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (le périmètre)*
4. `Surface`, *rôle : calcule la surface d'un disque de rayon r, entrée : 1 réel (le rayon), sortie : 1 réel (la surface)*
5. `Affiche_resultats`, *rôle : afficher les valeurs du périmètre et de la surface, entrée : 2 réels (périmètre et surface), sortie : vide,* **assuré par la fonction** `Ecrire()`

On obtient donc :

Lexique :{*principal*}
{*Les fonctions de saisie et d'affichage seront assurées par les fonctions* `Lire()` *et* `Ecrire()`}
{*Les deux fonctions à déclarer et à définir sont les fonctions de traitement.*}

{*R : Calcule le périmètre d'un cercle de rayon r*
E : 1 réel (le rayon)
S : 1 réel (le périmètre)}
`Perimetre` *: la fonction (r : 1 réel)→ 1 réel*

{*R : Calcule la surface d'un disque de rayon r*
E : 1 réel (le rayon)
S : 1 réel (la surface)}
`Surface` *: la fonction (r : 1 réel)→ 1 réel*

rayon, peri, surf : 3 réels

Algorithme :{*principal*}
Début
Ecrire("Ce programme vous permet de saisir le rayon d'un disque à partir duquel, le périmètre et la surface du disque sont calculés puis affichés")
Ecrire("Saisissez le rayon du disque en cm") {*on indique à l'utilisateur ce qu'on attend de lui*}
Lire(rayon) {*On sauvegarde sa saisie dans la variable locale rayon (déclarée dans le lexique)*}
peri⟵`Perimetre`*(rayon)*
surf⟵`Surface`*(rayon)*
Ecrire("Le périmètre de votre disque est :", peri, "cm")
Ecrire(" et sa surface est : ", surf, "cm²")
Fin

{*Définitions des fonctions*}
`Perimetre` *: la fonction (r : 1 réel)→ 1 réel*
Lexique :{*Local à* `Perimetre`}
p : 1 réel
PI : La constante réelle :=3.14159265
Algorithme :{*Local à* `Perimetre`}
Début

```
  p⟵2*PI*r
  Retourner p
 Fin

Surface : la fonction (r : 1 réel)⟶ 1 réel
 Lexique :{Local à Surface}
  s : 1 réel
  PI : La constante réelle :=3.14159265
 Algorithme :{Local à Surface}
 Début
  s⟵PI*r*r
  Retourner s
 Fin
```

Retour à l'énoncé : 3.3

Chapitre 4

Codage en langage C

Il vous est très fortement recommandé d'installer chez vous une version de Visual Studio. L'inscription à l'Université vous permet de bénéficier des partenariats entre Microsoft et l'Université de Saint-Etienne via la plateforme MSDN (Academic Alliance de Microsoft). Vous pouvez récupérer gratuitement et légalement :

— Des versions de Windows
— Des versions de Visual Studio
— Beaucoup d'autres logiciels Microsoft, dont la suite office365.

Vous êtes libre d'installer ou non ces logiciels, mais il est fortement recommandé de pratiquer la programmation en langage C également en dehors des TD pour bien vous entraîner et progresser.
Vous pouvez par exemple installer des versions gratuites : Visual Studio Community ou encore plus légères : Code::Blocks.

Contrairement aux exercices d'algorithmie, la plupart des exercices de codage en langage C ne sont pas corrigés sur ce fascicule. A vous d'essayer chez vous et de venir poser des questions lors des séances de CM ou de TD.

Méthode de travail à adopter en séance :

1. Lors des TDs, vous devez utiliser/chercher dans le cours et le tutoriel les informations dont vous avez besoin. Il est donc nécessaire d'avoir son cours et d'afficher à l'écran le tutoriel qui répond à 90% des questions que vous vous posez en séance.
2. Pour gagner du temps en séance de TD, vous pouvez créer un seul projet en début de séance, puis **à la fin** de chaque exercice, c'est à dire lorsque votre programme fonctionne correctement, vous devez :
   (a) Sauvegarder vos fichiers **sources (.cpp et .h)** et uniquement ceux-ci (sur `U:\` et éventuellement sur votre clé USB)
   Pour cela, sous visual studio, dans l'arborescence du projet, double cliquez sur chaque fichier .cpp ou .h pour les rendrent actifs (ie : contenu visible dans l'éditeur de texte), puis dans le menu fichier de l'IDE, cliquez sur "enregistrez-sous" et stockez le dans votre espace personnel organisé !
   (b) Exclure du projet ces fichiers :
   Pour cela, sous visual studio, dans l'arborescence du projet, clic droit sur le fichier à exclure > Exclure du projet

(c) Créer de nouveaux fichiers pour commencer l'exercice suivant.

(d) Eventuellement, dans le menu `Générer`, vous pouvez `Nettoyer la solution` afin d'effacer les fichiers temporaires (`.obj` entre autres) et le fichier `.exe` de l'exercice précédent.

3. Il est indispensable de commenter vos codes !
   A la fois pour nous quand nous corrigeons, mais aussi et surtout pour vous quand vous reprendrez votre travail 6 mois plus tard en vous redemandant pourquoi vous avez fait tel ou tel choix !
4. Enfin, il faut **tester** votre programme pour voir s'il fonctionne bien comme attendu.
   — Compiler n'est pas "tester"
   — Générer l'exécutable n'est pas "tester"
   — Constater que votre programme donne le bon résultat pour une valeur n'est pas "tester"
   — Tester c'est chercher à mettre son programme en défaut en regardant les cas limites, ou à le bombarder de vecteurs de tests générés aléatoirement afin de comparer ses récultats avec ceux attendus.
   Le site `iutenligne` propose un petit "test" pour savoir si vous êtes un bon programmeur, vous pouvez consulter cette ressource en cliquant sur le lien suivant : Suis-je un bon programmeur?

4.1 [Ch] Prise en main de Visual Studio

Suivre les tutoriels vidéos disponibles (Tuto1 à Tuto3) sur claroline connect pour :

1. Installer Visual Studio Community
2. Créer un premier projet, le compiler et exécuter le programme
3. Utiliser le débogueur.

— Tutoriel1: Installation Visual Studio
— Tutoriel2: Fin installation Visual et premier projet
— Tutoriel3: Projet sous visual et travail attendu en séance (sur vos PC ou sur un PC de la salle)

Dans les deux premiers tutoriels il y a eu un problème lors de la conversion des enregistrements en mp4. Les premières secondes montrent un écran grisé qui disparaît rapidement. Dans la deuxième vidéo, la fin est tronquée (limitation de 10 min avec le premier logiciel utilisé), mais l'essentiel de la fin est repris plus précisément dans le tutoriel 3.

4.2 [TD] Prise en main de Code::Blocks

1. Récupérer le tutoriel en version `.pdf` sur Claroline Connect.
2. Sur le disque local (C :\Temp) de l'ordinateur, créer un répertoire `Info1`
3. Suivre la première section du tutoriel, **Présentation & démarrage**.
   Remarque :

- une installation est disponible sur le répertoire réseau `APPLIS` (`X:\GEII\`). Vous pouvez donc le lancer directement depuis le réseau.
- vous pouvez aussi choisir d'installer vous-même **Code::Blocks EDU-Portable** sur une clé USB. Vous devrez donc avoir votre clé USB à chaque séance pour lancer **Code::Blocks**.

4. Suivre la seconde section du tutoriel, **Création d'un nouveau projet & premier programme** `HelloWorld`, pour créer le projet `essai` contenant un fichier `essai.cpp` permettant d'afficher à l'écran `Hello World!`.
   Vérifier que vous savez
   (a) Taper le code sans erreur
   (b) Compiler le code
   (c) Générer l'exécutable
   (d) Exécuter le programme
5. Sauvegarder uniquement votre fichier source (ici `essai.cpp`) sur votre espace de stockage (`U:`).
   Organiser votre espace de stockage sous forme de répertoires.
   (p.ex. `U:\Info1\MesProjets\TP1\Exo_prise_en_main_codeblocks\essai.cpp`)
6. Vérifiez que le fichier `essai.cpp` est bien à votre endroit de sauvegarde.
7. Suivez à présent la section du tutoriel **Débogage**.
8. Puis passez à l'exercice 4.4.

4.3 [TD] Débogage d'un code

On donne le code suivant (qui contient de nombreuses erreurs de syntaxe) dont le rôle est de permettre à l'utilisateur d'entrer deux entiers, d'afficher les deux entiers saisis par l'utilisateur puis de calculer et d'afficher leur somme et leur produit.

```
#include<iostream>

using namespace std;

int main()
{
int a,b;
cout<<"Entrer deux entiers:\n";
cin<<a,b;
cout<<"Vous avez entré les valeurs a et b\n";
c=a+B
cout<<"Leur somme est: ",c;
c=a*b
cout<<"Leur produit est: ",c;
        c=a/b
        cout<<"La valeur approchée de ",a,"/",b,"est :",c;
return 0;
}
```

1. Récupérer (ou recopier) ce code sur l'ENT (fichier `debog.cpp`), compiler-le et corriger les erreurs de syntaxe
2. Une fois la compilation effectuée sans erreur, générer l'exécutable puis exécuter ce programme et vérifier qu'il fonctionne comme attendu, sinon corriger les erreurs de conception.
3. On souhaiterait ajouter des instructions permettant d'obtenir le quotient de a par b, le reste dans la division de a par b puis une valeur approchée de la fraction $\frac{a}{b}$
   (a) Proposer ces instructions en algorithmie
   (b) les traduire en C et effectuer la compilation en corrigeant les éventuelles erreurs
   (c) générer l'exécutable, exécuter le programme **avec débogage** en mode pas à pas en mettant un point d'arrêt (vous pourrez vous référer au tutoriel)
   (d) vérifier étape par étape son bon fonctionnement. Au besoin, corriger les éventuelles erreurs et recommencer.

4.4 [TD] Traduction d'algorithmes

Reprendre des algorithmes proposés dans les exercices 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.12 du chapitre 2 et les **traduire** en C.
Méthode de travail (pour gagner du temps en séance) :

1. Création d'un seul projet pour ces traductions d'algorithme.
2. Le code de l'algorithme (2.5) sera par exemple tapé dans un fichier `Trad_Algo_hms_s`, celui de l'exercice 2.6 dans le fichier `Trad_Algo_moy`, celui de l'exercice 2.7 dans le fichier `Trad_Algo_f2e`, celui de l'exercice 2.8 dans le fichier `Trad_Algo_TVA`
3. Une fois le code d'un algorithme compilé, exécuté et validé par des tests, enregistrer le fichier sous votre U :\ (voir tutoriel). Puis cliquer droit sur le fichier dans Visual et cliquer sur `Exclure du projet`.
4. Créer un nouveau fichier pour l'algorithme suivant et recommencer.

4.5 [TD] Sans gravité

Un des objectifs de cet exercice est de manipuler, en C, les nombres écrits en notation scientifiques.

La loi de gravitation universelle stipule que deux corps A et B de masses (en kg) respectives M_A et M_B et distants de d mètres, s'attirent avec des forces de même valeur mais de sens opposé. La valeur de cette force est donnée en Newton par :

$$F_{A\to B} = G\frac{M_A M_B}{d^2} \quad \text{où} \quad G = 6.67384 \times 10^{-11} \text{ SI}$$

— Proposer un algorithme principal permettant à un utilisateur d'entrer les masses de deux corps, la valeur de la force exercée par l'un des corps sur l'autre et de calculer et afficher la distance qui sépare ces deux corps.

— Créer un projet Visual Studio ou Code::Blocks et traduire [1] cet algorithme en C.
— Tester votre algorithme en entrant les données suivantes [2] :
  — Masse de la Terre : 5.972×10^{24} kg
  — Masse de la lune : 7.3477×10^{22} kg
  — Force d'attraction exercée par la Terre sur la Lune : 1.95×10^{20} N

  et donner approximativement la valeur de la distance Terre-Lune en kms.

4.6 [TD] Opérateurs mathématiques complexes avec les fonctions déclarées dans `math.h`

Ecrire un programme qui déclare deux réels a et b, l'un ayant une valeur négative et l'autre positive. Tester et afficher le résultat des fonctions `floor(x)`, `ceil(x)`, `abs(x)` et `sqrt(x)` sur chacune des deux variables, et analyser leur rôle.
Note : vous devez inclure le fichier d'en-tête `math.h` pour pouvoir utiliser ces fonctions, grâce à l'instruction de pré-processeur `#include<math.h>`.
Cette instruction doit être placée en en-tête du fichier.

Pour aller plus loin : accédez au site `http://www.cplusplus.com/` et faites une recherche du fichier `math.h`. Vous aurez ainsi accès au panel de fonctions mathématiques déclarées dans ce fichier (par exemple les fonctions trigonométriques cosinus, sinus, tangente, mais aussi les fonctions puissance, logarithme,...). N'hésitez pas à "jouer" avec ces fonctions dans votre programme.

Remarque 2 *Pour utiliser la constante Pi telle que définie dans* `math.h`*, il faut écrire en en-tête de fichier :*

```
#define _USE_MATH_DEFINES
#include<math.h>
```

La constante s'utilise ensuite en tapant `M_PI`*.*

1. Vous devrez utiliser des fonctions qui ne font pas partie des opérateurs de base. L'inclusion de `#include<math.h>` permet de déclarer l'utilisation d'un grand nombre de fonctions mathématiques (puissance, sinus, cosinus, tangente, racine carrée, racine cubique, etc...). Une recherche sur internet peut vous être utile ici pour connaître le nom des fonctions et leurs prototypes.

2. Pour saisir un nombre réel en notation scientifique en C, il faut le déclarer en float ou double puis ecrire `3.5e-4` par exemple pour représenter 3.5×10^{-4}

Chapitre 5

Les fonctions en C

5.1 [TD] A partir du cours

Dans cet exercice, on propose de reprendre les codes des 3 fonctions vues en cours : `d_atan`, `Deg2Far` et `g` et créer les 3 projets sous Visual Studio en respectant le découpage en fichiers `.h`, `.cpp` et `main.cpp`

1. Pour chaque projet, on vous demande de :
   — Recopier le code dans les fichiers correspondants. N'oubliez pas d'ajouter les commentaires sur le rôle de la fonction, les paramètres d'entrée et le type de retour de la fonction.
   — Compiler les fichiers `.cpp` et corriger les erreurs éventuelles.
   Faut-il compiler le `.h` ? pourquoi ?
   — Générer l'exécutable et exécuter le programme **sans débogage** dans un premier temps.
   — Si le programme s'exécute sans erreur mais que les résultats attendus ne sont pas corrects, exécuter le programme avec debogage et utiliser les points d'arrêt pour vérifier le bon déroulement pas à pas du programme(cf : Tutoriel).
   — Enregistrer les fichiers sources uniquement et uniquement les fichiers sources sur le `U:\` (puis éventuellement, copier les sur votre clé USB)

5.2 [TD] Traduction de fonctions simples (*)

On se servira des algorithmes établis dans le chapitre 3 sur les fonctions. Pour chaque exercice, il faut **tester** la fonction dans un programme principal, c'est à dire vérifier que votre programme donne les valeurs attendues sur plusieurs valeurs de test.

1. Traduire l'algorithme de la fonction permettant de calculer le volume d'une sphère de rayon r passé en paramètre.
   Exemple de valeurs de test :
   — $r = 0$, $vol\_sphere = 0$
   — $r = 1$, $vol\_sphere \approx 4.189$
   — $r = 3$, $vol\_sphere \approx 113.097$
   — $r = 10.35$, $vol\_sphere \approx 4644.187$

2. Traduire l'algorithme de la fonction qui calcule, à partir du prix hors taxe unitaire d'un article, du nombre d'articles, du taux de TVA en pourcentage, le prix TTC total de la commande de ces articles.
   Proposer quelques valeurs de test.

5.3 [TD] Découpage fonctionnel

1. Traduire le découpage fonctionnel vu dans l'exercice 3.3 (sur la surface du disque et le périmètre du cercle) en C.
2. Ajouter les fonctions permettant de calculer le volume d'un cylindre et la distance parcourue par un vélo en fonction du nombre de tours effectués par la roue (cf : Exercice 3.6)

5.4 [TD] Durée d'ensoleillement (*)

— Traduire le découpage fonctionnel simple de l'exercice 3.7
— Tester votre programme !
  — en vérifiant par exemple qu'à l'équateur, quelque soit le moment de l'année la durée d'ensoleillement est de 12h (ce qui se voit sur la formule)
  — qu'en s'approchant des pôles (dans la limite de $\pm 66°$ à cause de la formule simplifiée), on a des durées d'ensoleillement soit nulles, soit maximales en fonction du solstice d'hiver/été pris au pôle Nord/Sud.

5.5 [TD] Heures-minutes-secondes (**)

1. Proposer un découpage fonctionnel simple permettant à un utilisateur d'entrer un temps en secondes et qui calcule puis affiche ce temps sous la forme hh :mm :ss.
2. Traduire ce découpage fonctionnel en C.
3. Tester.

5.6 [TD] Courbure de la terre et horizon visible (***)

Le point le plus loin (l'horizon) que peut voir un humain étant en bord de mer (les pieds dans l'eau, donc altitude 0m) dépend de la hauteur de ses yeux par rapport au niveau de la mer et de la courbure de la terre qui limite sa vision.
L'objectif de cet exercice est de proposer un programme permettant de calculer et d'afficher la distance maximale (en kilomètres) à laquelle est situé l'horizon qu'un humain peut voir avant d'être limité par la courbure de la Terre en fonction de la hauteur de ses yeux (en mètres) par rapport au niveau de la mer. Cette hauteur prend en compte la distance yeux-pieds de l'humain ainsi que la distance pieds-niveau de la mer (l'humain peut être sur une dune, un promontoir, une montagne surplombant la mer, ou les pieds dans l'eau...).

1. Proposez l'algorithme de la fonction dont le rôle est de calculer la distance de l'horizon (en km) en fonction de la hauteur des yeux de l'humain (en m)
2. Traduisez cette fonction en langage C.

3. Proposez un programme principal de test permettant de calculer et d'afficher la distance de l'horizon quand vous avez les pieds dans l'eau. Êtes-vous surpris du résultat ?

Généralisation : Proposez l'algorithme (puis la traduction en C) d'une fonction similaire permettant de calculer la distance maximum de visibilité d'un objet qui serait situé à une hauteur h du niveau de la mer (le sommet d'une montagne, le haut du mat d'un navire voguant à l'horizon, ...), toujours en fonction de la hauteur des yeux de l'observateur. Cette fonction devra faire appel à la fonction précédente. Proposez un programme de test.

Chapitre 6

L'analyse par cas

6.1 [Ch] Sur les conditions logiques

Les conditions logiques sont très importantes avec la structure `Si...Fin`Si. Elles permettent d'évaluer si une condition est vraie ou fausse et ainsi savoir quelle suite d'instructions il faut effectuer dans l'algorithme.
Elles ne servent pas uniquement dans ce chapitre, on les retrouve quand on effectue du traitement itératif (boucles). Enfin vous avez également étudié ces notions en Maths (Logique) et en SIN (Algèbre de Boole).

1. On suppose que x et y sont deux entiers définis dans le Lexique. Donner les complémentaires (ie : la négation) des affirmations suivantes :
   — $x < 10$
   — $x < 0$ OU $x > 20$
   — $x < 0$ ET $x > 20$
   — $x \neq 0$ OU $y < 0$
   — NON($x \neq 0$) ET ($y < 0$ OU $y > 20$)
2. On donne $x \leftarrow 10$, $y \leftarrow 5$.
   Parmis les affirmations précédentes, indiquez lesquelles sont vraies. Vérifiez à l'aide des affirmations contraires trouvées en 1) que vos résultats sont cohérents. (Si une affirmation est vraie son contraire doit être faux.)

voir la correction : 6.9

6.2 [Ch] Quelques applications directes

Proposer l'algorithme des fonctions suivantes ainsi que celui d'un programme principal permettant de tester ces fonctions :

1. `Max` dont le rôle est de retourner le plus grand des deux nombres passés en paramètres.
2. `Affiche_cr` dont le rôle est d'afficher les deux nombres passés en paramètres par ordre croissant. Ex. `Affiche_cr(4.7,-3.2)` doit afficher `-3.2`≤`4.7`.
3. `Est_pair` dont le rôle est de **retourner** VRAI si l'entier passé en paramètre est pair et FAUX sinon.

voir la correction : 6.10

6.3 [TD] Interpolation linéaire

L'idée de cet exercice est de reprendre l'exercice 2.10 est de proposer un **découpage fonctionnel simple** permettant de demander à un utilisateur de saisir l'abscisse x_C du point C et de lui afficher les coordonnées de l'interpolation. En particulier, vous devrez vérifier que l'abscisse x_C saisie est bien dans le bon rang (entre x_A et x_B) avant d'appeler la fonction de traitement. Si ce n'est pas le cas, un message d'erreur devra être retourné et le programme prendra fin.

1. Dans un Lexique principal, proposez votre découpage fonctionnel ;
2. Proposez l'algorithme principal permettant de résoudre le problème posé ;
3. Donnez la définition de chaque fonction intervenant dans votre découpage fonctionnel.

6.4 [TD] Taille génétique

1. Dans un lexique principal, proposer un découpage fonctionnel simple permettant d'afficher à une personne quelconque sa taille génétique. La taille génétique d'un individu est obtenue en calculant la moyenne des tailles de ses parents puis en ajoutant 6cm si l'individu est de sexe masculin ou en retranchant 6cm si l'individu est de sexe féminin. Évidemment cela donne une information de base sur la taille future d'un enfant comme fonction de la taille de ses parents avec une très forte variabilité due à l'environnement, l'alimentation, l'hygiène de vie, le sport, ...
2. Proposer l'algorithme principal permettant de résoudre ce problème ;
3. Donner ensuite la définition de chaque fonction intervenant dans votre découpage fonctionnel.

6.5 [TD] Conditions multiples (*)

1. Saisie :
   — Dans un lexique principal, proposer un découpage fonctionnel simple [1] permettant à un utilisateur de saisir un nombre entier entre 1 et 50 et qui lui affiche un message d'erreur si le nombre saisi n'est pas dans le bon intervalle.
   — Proposer l'algorithme principal permettant de résoudre le problème.
   — Donner les algorithmes de chacune des fonctions intervenant dans le découpage.
2. Signe produit :
   — Dans un lexique principal, proposer un découpage fonctionnel simple permettant à un utilisateur de saisir deux nombres et qui lui affiche le signe de leur produit **sans calculer le produit**
   — Proposer l'algorithme principal permettant de résoudre le problème.
   — Donner les algorithmes de chacune des fonctions intervenant dans le découpage.

1. Rappel : en particulier, on utilisera directement les fonctions `Ecrire()` et `Lire()` s'il n'y a pas besoin d'affichage ou de saisie particuliers

6.6 [TD] Imbrication des Si (**)

Pour chaque problème, proposer un découpage fonctionnel simple permettant de résoudre le problème posé. Donner l'algorithme principal permettant de résoudre le problème posé, puis donner les algorithmes de chacune des fonctions intervenant dans le découpage.

1. Saisie d'un nombre et affichage d'un des messages suivant :
   — "strictement positif"
   — "strictement négatif"
   — "nul"
2. Saisie de 3 nombres et affichage du plus petit des trois.
3. Saisie de 3 nombres et affichage de ces 3 nombres dans l'ordre croissant.
   Donner le nombre moyen de comparaisons effectuées par votre algorithme.
4. Saisie d'une année entre 1901 et 2099 puis :
   — affiche un message d'erreur si l'année n'est pas dans le bon intervalle.
   — indique si l'année saisie entre 1901 et 2099 est bissextile
5. Affichage du lot gagné à une tombola à partir de la saisie d'un numéro de ticket, sachant que les numéros gagnants sont :
   — le n°2505 qui gagne une TV 3D
   — les numéros multiples de 7 qui gagnent un Blueray sauf s'ils sont aussi multiples de 36 auquel cas ils gagnent une chaîne HiFi.

6.7 [Ch] Le Selon

1. Proposer l'algorithme d'une fonction dont le rôle est d'afficher le jour de la semaine à partir du numéro du jour (entre 1 et 7) passé en paramètre (1 pour lundi, ..., 7 pour dimanche).
   proposer un algorithme principal permettant de tester votre fonction.
2. Calculatrice :
   — Dans un lexique principal, proposer un découpage fonctionnel simple permettant de simuler une calculatrice en proposant à un utilisateur de choisir parmis les opérations :
     — addition
     — soustraction
     — multiplication
     — division

     de deux nombres saisis au clavier et qui affiche le résultat.
     (On prendra soin d'éviter une division par zéro).

   — Proposer l'algorithme principal permettant de résoudre le problème.
   — Donner les algorithmes de chacune des fonctions intervenant dans le découpage.

voir la correction : 6.11

6.8 [TD] Pour aller plus loin (**)

Pour chaque exercice, on donnera l'algorithme principal permettant de résoudre le problème posé puis on donnera la définition de chaque fonction du découpage.

1. Proposer un découpage fonctionnel simple permettant, à partir d'un trinôme du second degré ($ax^2 + bx + c$), d'afficher l'ensemble de ses racines dans $\mathbb{R}$ en fonction du signe du discriminant.
2. Proposer l'algorithme de la fonction dont le rôle est de retourner VRAI ssi l'année passée en paramètre est bissextile.
   Pour rappel : une année est bissextile si elle est divisible par 4.
   Toutefois les années séculaires ne sont pas bissextiles sauf si elles sont divisibles par 400 mais pas par 4000.
3. Proposer un découpage fonctionnel simple permettant, à partir d'une date saisie au format `jj:mm:aaaa`, d'afficher si c'est une date valide ou non. (par exemple : il n'y a pas de 31 avril, ni de 29 février 2003).

6.9 [Corr] Sur les conditions logiques

Les conditions logiques sont très importantes avec la structure `Si...FinSi`. Elles permettent d'évaluer si une condition est vraie ou fausse et ainsi savoir quelle suite d'instructions il faut effectuer dans l'algorithme.

1. On suppose que x et y sont deux entiers définis dans le Lexique. Donner les complémentaires (ie : la négation) des affirmations suivantes :
   — NON($x < 10$) $\Leftrightarrow$ $x \geq 10$ notez bien la présence du "ou égal"
   — NON($x < 0$ OU $x > 20$) $\Leftrightarrow$ $x \geq 0$ ET $x \leq 20$
   — NON($\underbrace{x < 0 \text{ ET } x > 20}_{\text{toujours FAUX}}$) $\Leftrightarrow$ $\underbrace{x \geq 0 \text{ OU } x \leq 20}_{\text{toujours VRAI}}$
   — NON($x \neq 0$ OU $y < 0$) $\Leftrightarrow$ $x = 0$ ET $y \geq 0$
   — NON(NON($x \neq 0$) ET ($y < 0$ OU $y > 20$)) $\Leftrightarrow$ ($x \neq 0$) OU NON($y < 0$ OU $y > 20$) $\Leftrightarrow$ $x \neq 0$ OU ($y \geq 0$ ET $y \leq 20$)
2. On donne $x \leftarrow 10$, $y \leftarrow 5$.
   Parmis les affirmations précédentes, indiquez lesquelles sont vraies. Vérifiez à l'aide des affirmations contraires trouvées en 1) que vos résultats sont cohérents. (Si une affirmation est vraie son contraire doit être faux.)

Affirmation	Evaluation	Décision
$x < 10$	$10 < 10$	FAUX
$x < 0$ OU $x > 20$	$\underbrace{10 < 0}_{\text{FAUX}}$ OU $\underbrace{10 > 20}_{\text{FAUX}}$	FAUX
$\underbrace{x < 0 \text{ ET } x > 20}_{\text{toujours FAUX}}$	inutile	FAUX
$x \neq 0$ OU $y < 0$	$\underbrace{10 \neq 0}_{\text{VRAI}}$ OU $\underbrace{5 < 0}_{\text{FAUX}}$	VRAI
NON$(x \neq 0)$ ET $(y < 0$ OU $y > 20)$	$\underbrace{\underbrace{\text{NON}\underbrace{(10 \neq 0)}_{\text{VRAI}}}_{\text{FAUX}} \text{ ET } (\underbrace{5 < 0}_{\text{FAUX}} \text{ OU } \underbrace{5 > 20}_{\text{FAUX}})}_{\text{FAUX}}}_{\text{FAUX}}$	FAUX

La vérification de la véracité de la condition contraire vous permet de voir si vous n'avez pas fait d'erreur dans son expression ou dans l'application des valeurs de x et y.

Retour à l'énoncé : 6.1

6.10 [Corr] Quelques applications directes

Proposer l'algorithme des fonctions suivantes ainsi que celui d'un programme principal permettant de tester ces fonctions :

Lexique :{principal}

{R : retourne le plus grand de deux nombres passés en paramètres
E : 2 réels correspondant à ces deux nombres
S : 1 réel correspondant au max des deux.}
`Max` : la fonction (`x` : 1 réel, `y` : 1 réel)→ 1 réel

{R : Affiche deux nombres passés en paramètre dans l'ordre croissant
E : 2 réels correspondant aux deux nombres à afficher
S : vide (c'est une fonction d'affichage, elle ne retourne rien)}
`Affiche_cr` : la fonction (`x` : 1 réel, `y` : 1 réel)→ vide

{R : Renseigne sur la parité d'un nombre **entier** et retourne VRAI s'il est pair et FAUX sinon.
E : 1 **entier** dont on cherche à connaître la parité.
S : 1 booléen contenant la valeur VRAI si l'entier est pair et FAUX sinon
Attention : avant d'écrire quoique ce soit, vérifiez bien que vous travaillez sur des nombres entiers. On ne peut pas répondre à la question "3.2 est-il pair ou impair..." }
`Est_pair` : la fonction (`n` : 1 entier)→ 1 booléen

{Déclaration des variables utilisées pour tester les fonctions dans l'algorithme principal}
xx,yy,zz : 3 réels {pour tester la fonction `Max` et la fonction `Affiche_cr`}
nn : 1 entier {pour tester la fonction `Est_pair`}
bb : 1 booléen

```
Algorithme :{principal}
Début
 Ecrire("Test de la fonction Max")
 Ecrire("Entrer deux réels en validant après chaque saisie")
 Lire(xx,yy)
 zz←Max(xx,yy)
 Ecrire("Max(",xx,",",yy,")=",zz)

 Ecrire("Test de la fonction Affiche_cr")
 Ecrire("Entrer deux réels en validant après chaque saisie")
 Lire(xx,yy)
 Affiche_cr(xx,yy)

 Ecrire("Test de la fonction Est_pair")
 Ecrire("Entrer un entier")
 Lire(nn)
 bb←Est_pair(nn)
 Si bb
  Alors
   Ecrire(nn,"est pair")
  Sinon
   Ecrire(nn,"est impair")
 FinSi
Fin
```

{
Plusieurs remarques sur ce dernier appel :

- on aurait pu écrire Si bb=VRAI, mais comme bb est un booléen, il a pour valeur VRAI ou FAUX. On peut l'utiliser directement en tant que condition.
- comme la fonction `Est_pair` retourne un booléen, on aurait même pu se passer de la variable bb en écrivant : Si `Est_pair(nn)` directement.
- Il est indispensable de procéder à un affichage conditionnel dans l'algorithme principal. En effet, on ne peut pas afficher un booléen directement. Aussi une instruction du type Ecrire(bb) n'a pas de sens et est à proscrire.

}

```
   {Définitions des fonctions utilisées}
 Max : la fonction (x : 1 réel, y : 1 réel)→ 1 réel
  Lexique :{Local à Max}
   m : 1 réel
  Algorithme :{Local à Max}
  Début
   Si(x<y)
    Alors {y est le plus grand}
     m←y
    Sinon {x est le plus grand (NON(x < y) ⇔x≥ y)}
     m←x
   FinSi
```

Retourner `m`
Fin
{Important : C'est l'affectation qui est conditionnelle : on affecte à `m`, soit `x` soit `y`. Dans les deux cas, on a une seule instruction "Retourner" à la fin de l'algorithme et on retourne `m`.}

`Affiche_cr` : la fonction (`x` : 1 réel, `y` : 1 réel)→ vide
Lexique :{Local à `Affiche_cr`}
{vide}
Algorithme :{Local à `Affiche_cr`}
Début
Si(`x`<`y`)
Alors {`y` est le plus grand}
Ecrire(x,"<",y)
Sinon {`x` est le plus grand ($\overline{x < y} \Leftrightarrow$`x` $\geq$ `y`)}
Ecrire(y,"$\leq$",x)
FinSi
Fin

`Est_pair` : la fonction (`n` : 1 entier)→ 1 booléen
Lexique :{Local à `Est_pair`}
`b` : 1 booléen
Algorithme :{Local à `Est_pair`}
Début
Si(`n` reste 2 = 0)
Alors {`n` est pair}
`b`← VRAI
Sinon {`n` est impair}
`b`← FAUX
FinSi
Retourner `b`
Fin
{Important : Comme précédemment, c'est l'affectation qui est conditionnelle : on affecte à `b`, soit VRAI soit FAUX. Dans les deux cas, on a une seule instruction "Retourner" et on retourne `b`.}

Retour à l'énoncé : 6.2

6.11 [Corr] Le Selon

1. Proposer l'algorithme d'une fonction dont le rôle est d'afficher le jour de la semaine à partir du numéro du jour (entre 1 et 7) passé en paramètre (1 pour lundi, ..., 7 pour dimanche).

   {Définition de la fonction `jour_semaine`}
   {R : Affiche le jour de la semaine selon le numéro du jour passé en paramètre (ex : lundi correspond à 1, mardi à 2, etc...)
   E : 1 entier compris entre 1 et 7
   S : vide (C'est une fonction d'affichage, elle ne retourne rien)}

`jour_semaine` : la fonction (`j` : 1 entier)→ vide
Lexique :{Local à `jour_semaine`}
{vide}
Algorithme :{Local à `jour_semaine`}
Début
Selon `j` {Aucune comparaison est faite ici. On indique la variable dont on souhaite regarder la valeur et on discute **selon** les différentes valeurs qu'elle peu prendre}
1 : Ecrire("Lundi")
2 : Ecrire("Mardi")
3 : Ecrire("Mercredi")
4 : Ecrire("Jeudi")
5 : Ecrire("Vendredi")
6 : Ecrire("Samedi")
7 : Ecrire("Dimanche")
FinSelon
Fin

proposer un algorithme principal permettant de tester votre fonction
Lexique :{principal}
{Déclaration de la fonction qui sera utilisée :}
{R : Affiche le jour de la semaine selon le numéro du jour passé en paramètre (ex : lundi correspond à 1, mardi à 2, etc...)
E : 1 entier compris entre 1 et 7
S : vide (C'est une fonction d'affichage, elle ne retourne rien)}
`jour_semaine` : la fonction (`j` : 1 entier)→ vide

jour : 1 entier

Algorithme :{principal}
Début
Ecrire("Entrer le numéro d'un jour entre 1 et 7") {On est obligé de faire confiance à l'utilisateur. On peut éventuellement prévoir un cas `autrement` dans la structure `Selon` de la fonction `jour_semaine`, en indiquant qu'il y a une erreur de saisie. Essayez de l'ajouter}
Lire(jour)
`jour_semaine`(jour) {Appel de la fonction}
Fin

2. Calculatrice :
   — Dans un lexique principal, proposer un découpage fonctionnel simple permettant de simuler une calculatrice en proposant à un utilisateur de choisir parmis les opérations :
     — addition
     — soustraction
     — multiplication
     — division

de deux nombres saisis au clavier et qui affiche le résultat.
(On prendra soin d'éviter une division par zéro).

Lexique :{principal}
{Il faut demander à l'utilisateur 3 informations, deux opérandes et un opérateur. Les deux opérandes seront des réels, l'opérateur (+,-,*,/) peut être représenté par un caractère. On utilisera, pour ces saisies, directement la fonction `Lire`.}

{R : Réalise, **selon** l'opérateur passé en paramètre, des additions, soustractions, multiplications ou divisions entre deux opérandes
E : 2 réels correspondant aux opérandes, 1 caractère correspondant au type d'opération à effectuer
S : 1 réel correspondant au résultat de l'opération }
`Calculatrice` : la fonction (`a` : 1 réel, `op` : 1 caractère, `b` : 1 réel)→ 1 réel

{On souhaite afficher le résultat de l'opération sous la forme `operande1 operateur operande2 = resultat`. Il faudra faire attention dans le cas d'une division par 0 et afficher un message d'erreur. C'est donc un affichage particulier, qui ne peut pas être réalisé directement par la fonction `Ecrire()`}
{R : Affiche le résultat de l'opération réalisée ou un message d'erreur dans le cas d'une division par 0.
E : 1 réel correspondant au résultat
S : vide}
`Affiche_resultat` : la fonction (result : 1 réel)→ vide

{Il y a un réel qui représente l'infini, nous allons donc introduire une constante réelle correspondant à l'infini. C'est cette valeur que retournera la fonction dans le cas d'une division par 0. (On verra dans le chapitre suivant comment traiter ce cas en C/C++).
Ce réel va être utilisé dans la fonction `Calculatrice` et dans la fonction `Affiche_resultat`, nous avons donc intérêt à le déclarer **GLOBALEMENT** dans le lexique principal.
}
INFTY : la constante réelle :=∞

`op1`, `op2`, `res` : 3 réels {on déclare les deux opérandes et le résultat}
`op` : 1 caractère {on déclare la variable qu'on va utiliser pour stocker le caractère correpondant à l'opération qu'on souhaite effectuer : '+', '-', '*' ou '/'}

— Proposer l'algorithme principal permettant de résoudre le problème.
Algorithme :{principal}
Début
Ecrire("Saisissez votre opération sous la forme : a opérateur b, puis validez")
Lire(`op1`,`op`,`op2`)
`res`←`Calculatrice(op1,op,op2)`
`Affiche_resultat(res)`
Fin
{On peut constater encore une fois que l'algorithme principal est très simple à

lire et à comprendre puisque tout le travail est effectué par des fonctions, dont on a bien pris soin de préciser le rôle.}

— Donner les algorithmes de chacune des fonctions intervenant dans le découpage. {Nous avons donc une fonction à définir dans un premier temps, la fonction `Calculatrice`. Une des difficultés va être de traiter le cas de la division par zéro. Notre fonction ne doit rien afficher (ce n'est pas son rôle) et doit retourner un réel dans tous les cas.}

`Calculatrice` : la fonction (`a` : 1 réel, `op` : 1 caractère, `b` : 1 réel)→ 1 réel
Lexique :{Local à `Calculatrice`}
`resultat` : 1 réel
Algorithme :{Local à `Calculatrice`}
Début
Selon `op` {On discute selon les valeurs du caractère `op` l'opération qu'on souhaite effectuer}
'+' : `resultat`← `a+b` {NB : les caractères se notent entre simple quote ' ', si vous les omettez le `+` est considéré comme l'opérateur d'addition et pas comme le caractère `'+'`}
'-' : `resultat`← `a-b`
'*' : `resultat`← `a*b`
'/' : Si `b=0` {on traite d'abord le cas particulier de la division par 0}
Alors
`resultat`←INFTY {Constante déclarée et initialisée GLOBALEMENT}
Sinon {`b`≠ 0 et on peut diviser par `b`}
`resultat`←`a/b`
FinSi
FinSelon
Retourner `resultat`
Fin

{Encore une fois, c'est l'**affectation** qui est conditionnelle. Il y a une seule instruction Retourner à la fin de la fonction.
D'autre part, on voit qu'on peut inclure une structure `Si...Alors...Sinon...FinSi` dans un `Selon`.}

`Affiche_resultat` : la fonction (result : 1 réel)→ vide
Lexique :{Local à `Affiche_resultat`}
{vide}
Algorithme :{Local à `Affiche_resultat`}
Début
Si `result`=INFTY {on traite d'abord le cas particulier}
Alors
Ecrire("Erreur : division par 0 impossible")
Sinon {On traite le cas général, `result` est un nombre réel contenant le résultat du calcul}
Ecrire("=",`result`)
FinSi
Fin

Retour à l'énoncé : 6.7

Chapitre 7

L'analyse par cas en C

Dans les deux exercices suivants, on vous demande de reprendre les algorithmes effectués lors du TD précédent et de les traduire en C. Ne réinventez pas les algorithmes et utilisez cours+tutoriel pour la création des projets, la traduction des notions algorithmiques en C, etc...

7.1 [TD] Traduction du Si en C avec des fonctions (*)

1. Proposer les traductions en C des algorithmes des fonctions suivantes ainsi que celle d'un programme principal permettant de tester ces fonctions :
   (a) `Max` dont le rôle est de retourner le plus grand des deux nombres passés en paramètres.
   (b) `Est_pair` dont le rôle est de retourner VRAI [1] si l'entier passé en paramètre est pair et FAUX sinon.
2. Traduire en C, le découpage fonctionnel simple permettant à un utilisateur de saisir deux nombres et de lui afficher le signe de leur produit **sans calculer le produit**.

7.2 [TD] Traduction du Selon en C avec des fonctions (**)

1. Traduire en C le découpage fonctionnel simple permettant de simuler une calculatrice en proposant à un utilisateur de choisir parmi les opérations :
   — addition
   — soustraction
   — multiplication
   — division
   de deux nombres saisis au clavier et qui affiche le résultat.
   Lors de la division, si le diviseur est nul, on pourra retourner l'infini.
   Pour ce faire, vous ajouterez en en-tête de fichier : `#include<limits>` puis dans le lexique de la fonction de traitement, vous déclarerez une constante réelle nommée `INF` qui sera initialisée avec la valeur : `std::numeric_limits<double>::infinity();`

1. On rappelle que le type booléen n'existe pas en C. Vous reportez au cours pour voir comment on représente FAUX en C.

7.3 [TD] Algo + C (**)

1. Proposer un découpage fonctionnel simple permettant d'afficher le lot gagné à une tombola à partir de la saisie d'un numéro de ticket, sachant que les numéros gagnants sont :
   — le n°2505 qui gagne une TV 3D
   — les numéros multiples de 7 qui gagnent un CD sauf s'ils sont aussi multiples de 36 auquel cas ils gagnent un DVD.
2. Proposer une traduction en C de ce découpage fonctionnel.

Chapitre 8

Les structures itératives

8.1 [Ch] Boucles élémentaires

1. Proposer l'algorithme d'une fonction dont le rôle est de calculer la somme des carrés des entiers naturels de 1 à n.
   Proposer un algorithme principal permettant de tester votre fonction.
2. Proposer l'algorithme de la fonction `factorielle` dont le rôle est de calculer le produit des entiers naturels de 1 à n.
   Proposer un algorithme principal permettant de tester cette fonction.

Voir la correction : 8.8

8.2 [TD] Séance de tirs au but (**)

On se propose de réaliser un jeu de tirs au but entre deux équipes. Tour à tour, chaque équipe doit tenter de marquer un but. On comptabilise petit à petit le score des équipes. Dès qu'une équipe atteint au moins six buts avec nécessairement deux buts d'écart, elle remporte la victoire.

La simulation d'un tir au but suivra la procédure suivante :

1. génération d'un entier aléatoire.
2. saisie d'un entier par l'utilisateur.
3. résultat du tir : si les deux entiers ont la même parité, le but est marqué, sinon c'est manqué.

Proposer l'algorithme qui va réaliser ce jeu, en suivant ces étapes :

1. Ecrire l'algorithme principal du jeu.
   - La simulation d'un tir au but doit faire l'objet d'une fonction qui sera déclarée dans le lexique principal.
2. Décrire la fonction qui va réaliser la simulation d'un tir au but.
   - Lors de sa description, cette fonction déclarera dans son lexique local une fonction nommée `rand` qui ne prend pas d'argument et renvoie une valeur entière aléatoire (on ne s'intéressera pas à la description de cette fonction).
   - Pour déterminer la parité des entiers, on pourra utiliser la fonction `Est_pair` développée dans l'exercice 6.2.

8.3 [TD] Racine carrée (**)

1. Proposer l'algorithme d'une fonction dont le rôle est de calculer une approximation à l'ordre n de la racine carrée d'un nombre réel positif par l'algorithme de Newton.
   Principe : Pour tout $a \in \mathbb{R}_+$, la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$\begin{cases} u_0 & = & 1 \\ u_{n+1} & = & \frac{u_n + \frac{a}{u_n}}{2} \end{cases}$$

   converge vers $\sqrt{a}$
2. Proposer l'algorithme principal permettant d'appeler votre fonction.

8.4 [TD] Suite de Fibonacci (**)

1. Proposer l'algorithme d'une fonction dont le rôle est de calculer le terme d'indice n (c'est à dire le $n+1^e$ terme) de la suite de Fibonacci initialisée par les deux entiers a et b et définie par :

$$\begin{cases} u_0 & = & a \\ u_1 & = & b \\ u_n & = & u_{n-1} + u_{n-2} \text{ pour } n \geq 2 \end{cases}$$

2. Proposer l'algorithme principal permettant d'appeler votre fonction.

8.5 [TD] PGCD (**)

1. Proposer l'algorithme d'une fonction dont le rôle est de calculer le plus grand commun diviseur de deux nombres entiers positifs par l'algorithme d'Euclide.
   Principe (récursif) :

$$\texttt{pgcd}(a,b) = \begin{cases} a & \text{Si } b = 0 \\ \texttt{pgcd}(b, a \text{ reste } b) & \text{sinon} \end{cases}$$

   Conseil : Effectuer un exemple avec $a = 74$ et $b = 6$ par exemple pour déterminer la structure **itérative** à utiliser.
2. Tester votre fonction dans un algorithme principal dans lequel vous demanderez deux entiers à l'utilisateur et lui afficherez leur pgcd.

8.6 [TD] Tables de multiplication (**)

1. Proposer l'algorithme d'une fonction dont le rôle est d'afficher la table de Pythagore de la multiplication :

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

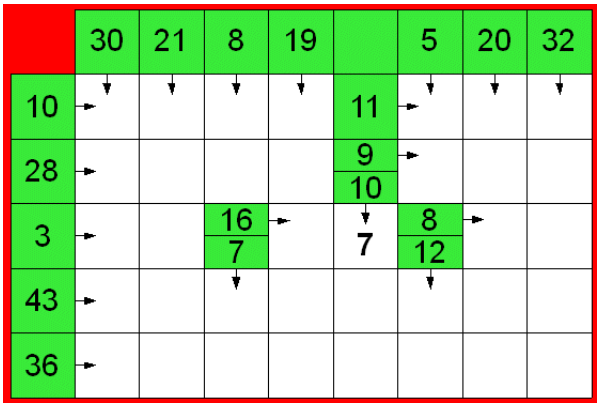
2. Proposer l'algorithme principal permettant d'appeler votre fonction.

8.7 [TD] Kakuro : nombres fléchés (***)

Le Kakuro [1] est un jeu japonais ancien qui est aux chiffres ce que les mots flechés sont aux lettres.
L'objectif du jeu est de remplir les cases vides avec des chiffres de 1 à 9 de sorte que la somme des chiffres mis dans les cases vides soit égale au nombre inscrit dans la case remplie en suivant les règles suivantes :

- Chaque case vide doit contenir un chiffre entre 1 et 9
- Le nombre indiqué dans une case verte est égal à la somme des chiffres contenus dans les cases indiquées par la flèche.
- On ne peut pas utiliser plusieurs fois le même chiffre dans une même somme. Ainsi sur trois cases, 9 peut se décomposer en 6+2+1 mais pas en 7+1+1.
- Si deux nombres dans des cases vertes sont identiques, les décompositions doivent être différentes. Ainsi si deux 9 sont à décomposer sur 3 cases et qu'on a une décomposition en 6+2+1 pour l'un, on ne pourra avoir pour l'autre que 5+3+1 ou 4+3+2 mais pas 6+2+1.

Une grille (à remplir) est donnée à titre d'exemple dans la figure 8.7.
Pour aider à remplir cette grille, il serait bien d'avoir une liste des décompositions des



nombres en 2 cases, 3 cases, etc. . ..

1. D'après https://www.ilemaths.net/sujet-enigmo-21-nombres-fleches-210177.html

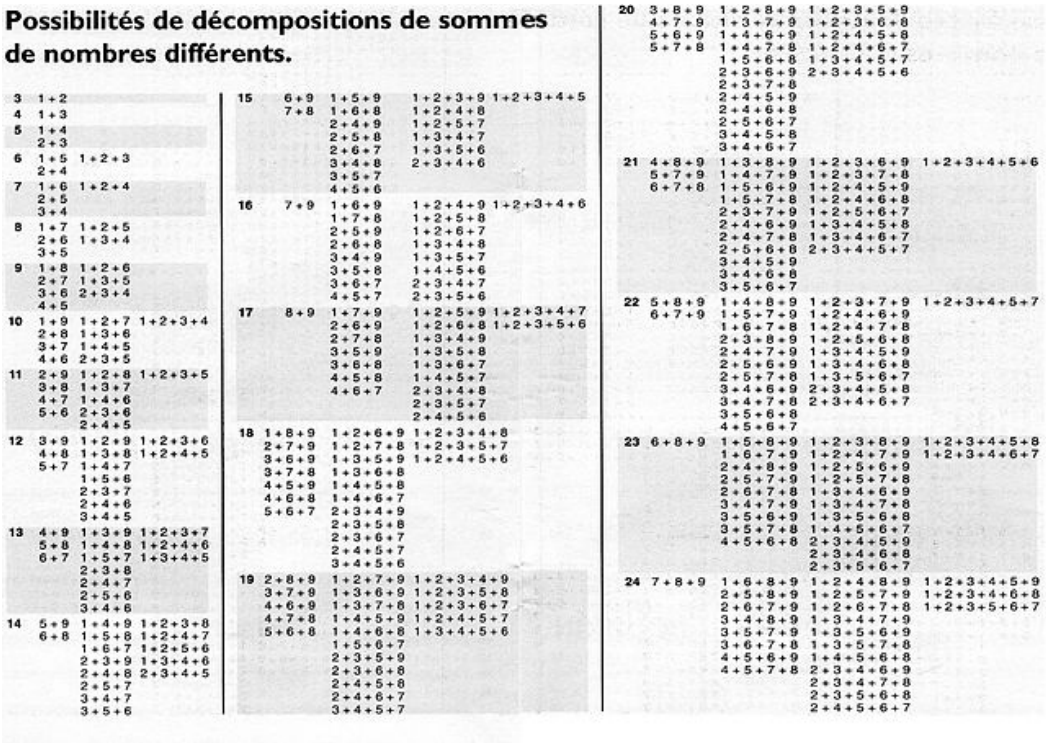
L'objectif de cet exercice est de proposer un programme qui permette de générer une liste complète de tous les nombres décomposables de toutes les manières possibles.

1. Combien y a-t-il de nombres décomposables (en suivant les règles du jeu), donnez le plus petit et le plus grand nombre décomposable.
2. Proposez l'algorithme d'une fonction dont le rôle est d'afficher la liste de toutes les décompositions possibles d'un nombre passé en paramètre en un nombre de cases donné.
   Exemple : pour 9 en 2, la fonction devra afficher

   ```
   Décomposition de 9 en 2:
   1+8
   2+7
   3+6
   4+5
   ```

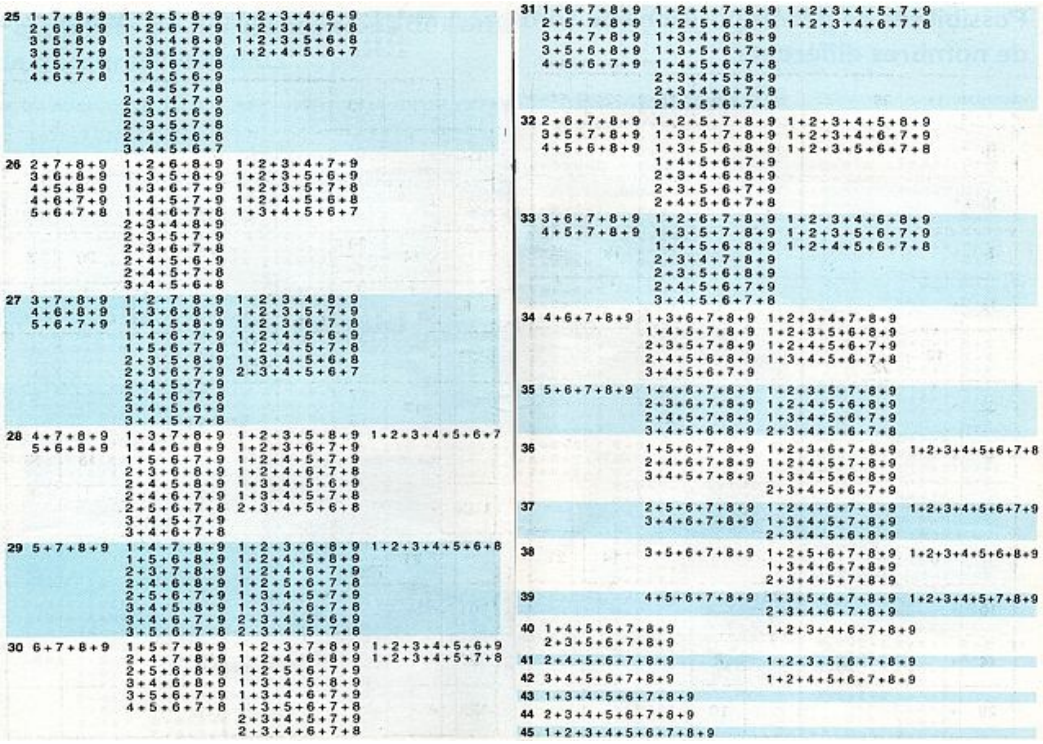
3. Proposez ensuite l'algorithme d'une fonction permettant, en utilisant la fonction précédente, d'afficher la liste de toutes les décompositions possibles. Cette fonction aura la particularité de ne prendre aucun paramètre d'entrée et ne retournera rien.

Afin de vérifier le bon fonctionnement de vos programmes, voici la liste exhaustive de toutes les décompositions possibles (répondant ainsi à la question 1).



Possibilités de décompositions de sommes de nombres différents.

3	1+2			
4	1+3			
5	1+4	2+3		
6	1+5	2+4	1+2+3	
7	1+6	2+5	3+4	1+2+4
8	1+7	2+6	3+5	1+2+5 1+3+4
9	1+8	2+7	3+6	4+5 1+2+6 1+3+5 2+3+4
10	1+9	2+8	3+7	4+6 1+2+7 1+3+6 1+4+5 2+3+5 1+2+3+4
11	2+9	3+8	4+7	5+6 1+2+8 1+3+7 1+4+6 2+3+6 2+4+5 1+2+3+5
12	3+9	4+8	5+7	1+2+9 1+3+8 1+4+7 1+5+6 2+3+7 2+4+6 3+4+5 1+2+3+6 1+2+4+5
13	4+9	5+8	6+7	1+3+9 1+4+8 1+5+7 2+3+8 2+4+7 2+5+6 3+4+6 1+2+3+7 1+2+4+6 1+3+4+5
14	5+9	6+8		1+4+9 1+5+8 1+6+7 2+3+9 2+4+8 2+5+7 3+4+7 3+5+6 1+2+3+8 1+2+4+7 1+2+5+6 1+3+4+6 2+3+4+5
15	6+9	7+8		1+5+9 1+6+8 2+4+9 2+5+8 2+6+7 3+4+8 3+5+7 4+5+6 1+2+3+9 1+2+4+8 1+2+5+7 1+3+4+7 1+3+5+6 2+3+4+6 1+2+3+4+5
16	7+9			1+6+9 1+7+8 2+5+9 2+6+8 3+4+9 3+5+8 3+6+7 4+5+7 1+2+4+9 1+2+5+8 1+2+6+7 1+3+4+8 1+3+5+7 1+4+5+6 2+3+4+7 2+3+5+6 1+2+3+4+6
17	8+9			1+7+9 2+6+9 2+7+8 3+5+9 3+6+8 4+5+8 4+6+7 1+2+5+9 1+2+6+8 1+3+4+9 1+3+5+8 1+3+6+7 1+4+5+7 2+3+4+8 2+3+5+7 2+4+5+6 1+2+3+4+7 1+2+3+5+6
18	1+8+9	2+7+9	3+6+9	3+7+8 4+5+9 4+6+8 5+6+7 1+2+6+9 1+2+7+8 1+3+5+9 1+3+6+8 1+4+5+8 1+4+6+7 2+3+4+9 2+3+5+8 2+3+6+7 2+4+5+7 3+4+5+6 1+2+3+4+8 1+2+3+5+7 1+2+4+5+6
19	2+8+9	3+7+9	4+6+9	4+7+8 5+6+8 1+2+7+9 1+3+6+9 1+3+7+8 1+4+5+9 1+4+6+8 1+5+6+7 2+3+5+9 2+3+6+8 2+4+5+8 2+4+6+7 3+4+5+7 1+2+3+4+9 1+2+3+5+8 1+2+3+6+7 1+2+4+5+7 1+3+4+5+6
20	3+8+9	4+7+9	5+6+9	5+7+8 1+2+8+9 1+3+7+9 1+4+6+9 1+4+7+8 1+5+6+8 2+3+6+9 2+3+7+8 2+4+5+9 2+4+6+8 2+5+6+7 3+4+5+8 3+4+6+7 1+2+3+5+9 1+2+3+6+8 1+2+4+5+8 1+2+4+6+7 1+3+4+5+7 2+3+4+5+6
21	4+8+9	5+7+9	6+7+8	1+3+8+9 1+4+7+9 1+5+6+9 1+5+7+8 2+3+7+9 2+4+6+9 2+4+7+8 2+5+6+8 3+4+5+9 3+4+6+8 3+5+6+7 1+2+3+6+9 1+2+3+7+8 1+2+4+5+9 1+2+4+6+8 1+2+5+6+7 1+3+4+5+8 1+3+4+6+7 2+3+4+5+7 1+2+3+4+5+6
22	5+8+9	6+7+9		1+4+8+9 1+5+7+9 1+6+7+8 2+3+8+9 2+4+7+9 2+5+6+9 2+5+7+8 3+4+6+9 3+4+7+8 3+5+6+8 4+5+6+7 1+2+3+7+9 1+2+4+6+9 1+2+4+7+8 1+2+5+6+8 1+3+4+5+9 1+3+4+6+8 1+3+5+6+7 2+3+4+5+8 2+3+4+6+7 1+2+3+4+5+7
23	6+8+9			1+5+8+9 1+6+7+9 2+4+8+9 2+5+7+9 2+6+7+8 3+4+7+9 3+5+6+9 3+5+7+8 4+5+6+8 1+2+3+8+9 1+2+4+7+9 1+2+5+6+9 1+2+5+7+8 1+3+4+6+9 1+3+4+7+8 1+3+5+6+8 1+4+5+6+7 2+3+4+5+9 2+3+4+6+8 2+3+5+6+7 1+2+3+4+5+8 1+2+3+4+6+7
24	7+8+9			1+6+8+9 2+5+8+9 2+6+7+9 3+4+8+9 3+5+7+9 3+6+7+8 4+5+6+9 4+5+7+8 1+2+4+8+9 1+2+5+7+9 1+2+6+7+8 1+3+4+7+9 1+3+5+6+9 1+3+5+7+8 1+4+5+6+8 2+3+4+6+9 2+3+4+7+8 2+3+5+6+8 2+4+5+6+7 1+2+3+4+5+9 1+2+3+4+6+8 1+2+3+5+6+7



8.8 [Corr] Boucles élémentaires

1. Proposer l'algorithme d'une fonction dont le rôle est de calculer la somme des carrés des entiers naturels de 1 à n.
   Proposer un algorithme principal permettant de tester votre fonction.
2. Proposer l'algorithme de la fonction `factorielle` dont le rôle est de calculer le produit des entiers naturels de 1 à n.
   Proposer un algorithme principal permettant de tester cette fonction.

{Nous faisons un seul algorithme principal pour tester les deux fonctions de l'exercice}
Lexique {principal}
{
R : Calcule et retourne la somme $S = 1^2 + 2^2 + \cdots + n^2$
E : L'entier n de la formule précédente
S : Un entier correspondant à la somme S
}
Somme_carres : la fonction(n : 1 entier)$\longrightarrow$ 1 entier

{
R : Calcule et retourne le produit $p = 1 \times 2 \times \cdots \times n$
E : L'entier n de la formule précédente
S : Un entier p correspondant au produit des n premiers entiers naturels non nuls
}
Factorielle : la fonction(n : 1 entier)$\longrightarrow$ 1 entier

nn, s, p : 3 entiers

Algorithme {principal}
Début
  Ecrire("Ce programme calcule la somme des carrés des n premiers naturels")
  Ecrire("Entrer n :")
  Lire(nn)
  $s \leftarrow$ Somme_carres(nn)
  Ecrire("La somme $1^2 + ...+$",nn,"^{2} vaut :",s)

  Ecrire("Ce programme calcule maintenant n!")
  Ecrire("Entrer n :")
  Lire(nn)
  $p \leftarrow$ Factorielle(n)
  Ecrire(n," !=",p)
Fin

{Définitions des fonctions}
Somme_carres : la fonction(n : 1 entier)$\longrightarrow$ 1 entier
  Lexique {local}
   s : 1 entier
   i : 1 entier

  Algorithme {local}
  Début
   {conditions d'initialisation}
   $i \leftarrow 1$ {on se place sur le premier entier}
   $s \leftarrow 0$ {la somme au départ est nulle}
   Tant que $i \leq n$
    $s \leftarrow s + i^2$ {on ajoute i^2 à la somme précédente}
    $i \leftarrow i + 1$ {on passe à l'entier suivant}
   Fin Tant que {on passe une dernière fois dans la boucle pour l'entier i=n}
   Retourner s
  Fin

Factorielle : la fonction(n : 1 entier)$\longrightarrow$ 1 entier
  Lexique {local}
   p : 1entier
   i : 1entier

  Algorithme {local}
  Début
   {conditions d'initialisation}
   $i \leftarrow 1$ {on se place sur le premier entier}
   $p \leftarrow 1$ {le produit au départ vaut 1}
   Tant que $i \leq n$
    $p \leftarrow p * i$ {on multiplie le produit précédent par le nouvel entier}
    $i \leftarrow i + 1$ {on passe à l'entier suivant}
   Fin Tant que {on passe une dernière fois dans la boucle pour l'entier i=n}

Retourner p
Fin
{Quel est le résultat attendu lorsque $n = 0$? lorsque $n = 1$? Est-ce cohérent avec le résultat attendu quand on cherche 0!, 1! ?}

Retour à l'énoncé : 8.1

Chapitre 9

Les structures itératives en C

Pour les exercices suivants, reprendre les algorithmes effectués dans le chapitre précédent et les traduire ! (ne surtout pas les réinventer...)

9.1 [TD] PGCD (**)

1. Traduire l'algorithme de la fonction dont le rôle est de calculer le plus grand commun diviseur de deux nombres entiers positifs par l'algorithme d'Euclide.
   Principe (récursif) :

$$\texttt{pgcd}(a,b) = \begin{cases} a & \text{Si } b = 0 \\ \texttt{pgcd}(b, a \text{ reste } b) & \text{sinon} \end{cases}$$

2. Tester votre fonction dans un algorithme principal dans lequel vous demanderez deux entiers à l'utilisateur et lui afficherez leur pgcd.
   Conseil : Vérifiez que vous retrouvez le même résultat sur l'exemple fait dans le chapitre précédent avec $a = 74$ et $b = 6$.

9.2 [TD] Séance de tirs au but (**)

Coder l'algorithme du jeu de tirs au but développé dans l'exercice 8.2.
Aide : Pour générer un entier aléatoire compris entre m et n, il faut rajouter les lignes de code suivantes :

```
#include<time.h> //inclusion de la fonction time()
#include<stdlib.h> //inclusion des fonctions rand() et srand()
//puis dans la fonction où vous voulez générer l'entier aléatoire:
unsigned int alea;
srand(time(NULL)); //initialisation du générateur
alea=(rand() % (n-m+1))+m; //génére un entier aléatoire entier compris
                           //entre m et n (avec m<n)
```

9.3 [TD] Racine carrée (**)

1. Traduire l'algorithme de la fonction dont le rôle est de calculer une approximation à l'ordre n de la racine carrée d'un nombre réel positif par l'algorithme de Newton.
   Principe : Pour tout $a \in \mathbb{R}_+$, la suite $(u_n)_{n\in\mathbb{N}}$ définie par
   $$\begin{cases} u_0 & = & 1 \\ u_{n+1} & = & \frac{u_n + \frac{a}{u_n}}{2} \end{cases}$$
   converge vers $\sqrt{a}$
2. Testez votre fonction dans un algorithme principal.

3. A partir de quelles valeurs de n les approximations semblent-elles correctes ?
4. En déduire une estimation intuitive de l'efficacité de cet algorithme dans le calcul approché de $\sqrt{a}$.

9.4 [TD] Suite de Fibonacci (**)

1. Traduire l'algorithme de la fonction dont le rôle est de calculer le terme d'indice n (c'est à dire le $n+1^e$ terme) de la suite de Fibonacci initialisée par les deux entiers a et b et définie par :
   $$\begin{cases} u_0 & = & a \\ u_1 & = & b \\ u_n & = & u_{n-1} + u_{n-2} \text{ pour } n \geq 2 \end{cases}$$
2. Tester votre fonction dans un algorithme principal.

9.5 [TD] Tables de multiplication (**)

1. Traduire l'algorithme [1] de la fonction dont le rôle est d'afficher la table de Pythagore de la multiplication :

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

1. Il est possible d'aligner des nombres de différentes tailles en affichant sur une largeur commune prédéfinie et éventuellement de rajouter des 0 en poids forts grace aux instructions . setw() et setfill(). Ainsi : `cout << setw(5) << setfill('0') << 1;` affichera 00001. Pour que le compilateur connaisse setw et setfill il faudra inclure le fichier `iomanip` : `#include<iomanip>`

2. Testez votre fonction dans l'algorithme principal.

Les exercices suivants sont nouveaux et demandent une étude algorithmique avant de passer au codage.

9.6 [TD] Suite de Syracuse (***)

La suite de syracuse est définie de la manière suivante :

$$\begin{cases} u_0 & = & N \text{ Pour } N \in \mathbb{N}^\star \\ u_{n+1} & = & \begin{cases} \frac{u_n}{2} & \text{Si } u_n \text{ est pair} \\ 3u_n + 1 & \text{Si } u_n \text{ est impair} \end{cases} \end{cases}$$

Cette suite a donné lieu à une conjecture, appelée conjecture de Syracuse.
La conjecture stipule que quelque soit l'entier N de départ, la suite atteint toujours le cycle 4,2,1.

Cette conjecture n'a, à ce jour, pas été démontrée. [2]
En revanche, elle a été vérifiée informatiquement jusqu'à de très grands entiers.

1. Vérifier qu'à partir de $u_n = 1$, le cycle se répète.
2. Proposer l'algorithme d'une fonction dont le rôle est de calculer le terme u_n de la suite de Syracuse en fonction de n.
3. Proposer l'algorithme d'une fonction permettant de calculer le temps de vol défini par $tv = \min\{n \in \mathbb{N} \mid u_n = 1\}$ [3]
4. Proposer l'algorithme d'une fonction permettant de calculer l'altitude maximale de la suite, définie par : $alt = max\{u_n \mid n \in \mathbb{N}\}$ [4]
5. Proposer l'algorithme d'une fonction dont le rôle est d'afficher la suite de Syracuse d'un entier N jusqu'à la première apparition du cycle 4,2,1.
6. Proposer un algorithme principal permettant de tester vos fonctions
7. Traduire tous ces algorithmes dans un projet `Syracuse` comportant un fichier `Syracuse.h`, `Syracuse.cpp` et `main.cpp` et tester.

9.7 [TD] Dessins : motifs (**)

Proposer les fonctions permettant d'afficher les motifs suivants (le nombre de ligne doit être un paramètre des fonctions). Les fonctions seront testées successivement dans un algorithme principal.

1. Les `.` désignent un certain nombre de `*`

2. Paul Erdös, un des plus grands mathématiciens du XXe siècle a même affirmé que "Les mathématiques ne sont pas encore prêtes pour de tels problèmes", ce qui laisse songeur tant l'énoncé de ce problème est simple.
3. C'est le plus petit indice n tel que $u_n = 1$.
4. C'est la plus grande valeur u_n atteinte par la suite.

```
*
**
***
****
.   .
.    .
.     .
****...*
*****...*
******...*
```

2. Les . désignent un certain nombre de *

```
******...*
*****...*
****...*
.     .
.    .
.   .
****
***
**
*
```

3. Les . désignent un certain nombre de *

```
      *
     ***
    *****
   .......
  *.......*
 **.......**
***.......***
     |_|
```

4. Variante : on peut aussi ajouter des boules aléatoirement sur le sapin en proposant une fonction dont le rôle est le suivant :

```
Rôle: Lors de la génération du sapin, cette fonction est appelée pour afficher
une boule 'o' au lieu des '*' sur une ligne du sapin dont l'indice
est passé en paramètre.
Entrée: l'indice de la ligne considérée: 1 entier
        Les indices des emplacements libres: 1 tableau d'entiers
Sortie: l'emplacement de la boule sur la ligne considérée: 1 entier
```

Remarque : la boule doit être sur le sapin, autrement dit l'entier retourné par la fonction doit être compris entre les indices des extrémités du sapin !
Ensuite, cette fonction peut être appelée plusieurs fois pour placer un certain nombre de boules sur le sapin.

A cette fin, le morceau de code suivant peut être utile.
Pour générer un entier aléatoire compris entre m et n, il faut rajouter les lignes de code suivantes :

```
#include<time.h> //inclusion de la fonction time()
#include<stdlib.h> //inclusion des fonctions rand() et srand()
//puis dans la fonction où vous voulez générer l'entier aléatoire:
unsigned int alea;
srand(time(NULL)); //initialisation du générateur
alea=(rand() % (n-m+1))+m; //génére un entier aléatoire entier compris
                           //entre m et n (avec m<n)
```

9.8 [TD] Dessins : ensemble de points (**)

Proposer une fonction permettant de dessiner l'ensemble suivant :

$$\mathcal{E} = \{(x, y) \in \mathbb{R}^2 | 2(x-2)^2 + 5(y-3)^2 = 8\}$$

On pourra afficher ␣* si le point appartient à $\mathcal{E}$ et ␣␣ sinon.
Il faudra définir les variations sur x et sur y.

9.9 [TD] Défi Kangourou (****)

Ce problème est tiré d'un exercice du Kangourou des mathématiques destiné aux élèves de 5^e.
La formulation originale pour les élèves de 5^e est la suivante :
On considère une boîte parallélépipèdique rectangle (un pavé droit), aux dimensions **entières** *(c'est à dire que la largeur l, la longueur L et la hauteur h de la boîte sont des entiers naturels).*
On souhaite trouver les dimensions L, l et h d'une boîte dont le volume doit être plus grand qu'un volume minimum V_{min} donné et dont la somme des surfaces des 6 faces est la plus petite possible.

Niveau Kangourou : Donnez la réponse pour $V_{min} = 1992$

Niveau supérieur : Si avec un simple programme informatique étudiant tous les cas possibles, il est facile de répondre à cette question il faut noter que :

(a) Les élèves de 5^e n'avaient pas accès à l'outil informatique

(b) Lorsque le volume V_{min} augmente, il devient compliqué (voire impossible en un temps humainement raisonnable) d'étudier toutes les dimensions possibles pour trouver celles qui minimisent la surface pour un volume minimal donné.

Le problème auquel nous nous intéressons ici est de répondre à la question initiale **MAIS** pour des volumes V_{min} de plus en plus grands.
Pour représenter des grands entiers, on pourra utiliser le type `unsigned long long int` qui permet de coder des entiers sur 64 bits (entre 0 et $2^{64} - 1 = 18446744073709551615$). Ils se manipulent comme des `unsigned int` dans le code. Pour que la valeur du volume soit interprétée par le compilateur comme un unsigned long long int, il faudra rajouter lors de l'affectation du volume dans `Vmin` le

suffixe `llu`.
Exemple : `unsigned long long int Vmin=17897938237867950080llu;`
Vous testerez votre code (et son efficacité) sur les volumes `Vmin` suivants.

1992	6860	93151
960499	9708967	93989351
998047937	9731810376	98291308375
996305088416	9920820343510	99768224300572
998281001279379	9994319468547622	99896698153642460
997099808269056699	9976100633400818888	18439010748097173354
1772703	88263842051653632	130077551459565568
1035827914295214080	17897938237867950080	

Pour chaque volume, vous devez donner `L`, `l` et `h` ainsi que la somme `surface_min` des surfaces des 6 faces et le volume `V` occupé par la boîte :

V_{min}	L	l	h	S_{min}
1992				
6860				
93151				
960499				
9708967				
93989351				
998047937				
9731810376				
98291308375				
996305088416				
9920820343510				
99768224300572				
998281001279379				
9994319468547622				
99896698153642460				
997099808269056699				
9976100633400818888				
18439010748097173354				
1772703				
88263842051653632				
130077551459565568				
1035827914295214080				
17897938237867950080				

Chapitre 10

Les tableaux

10.1 [Ch] A partir d'algorithmes donnés

Lexique1 :
`taille` : 1 entier
`tab` : 1 tableau de `taille` réel

Algorithme1 :
Début
`tab[0]`⟵3.4 {1}
`tab[taille-1]`⟵`tab[0]`-1 {2}
Ecrire("Le premier élément de ce tableau est : ",`tab[1]`) {3}
Ecrire("Le dernier élément de ce tableau est : ",`tab[taille]`) {4}
Fin

Lexique2 :
`tab` : 1 tableau de 6 réel :={-4.2,5.4,8.4,9.3,-4.5,0,4.3,-2.5}
`tab2` : 1 tableau de 6 réel :={-3.4,5.2,1.9}
`i` : un réel

Algorithme2 :
Début
Ecrire("Saisir un indice pour voir quel est le i^e élément du tableau") {1}
Lire(`i`) {2}
Ecrire("Le i^e élément de ce tableau est : ",`tab[i]`) {3}
Ecrire("Quel est l'indice de la case que vous souhaitez modifier et quelle valeur souhaitez vous mettre ?") {4}
Lire(`i`,`val`) {5}
`tab[i+1]`⟵`val` {6}
Ecrire("Vous avez modifié la, ",i,"^{e} case. Son nouveau contenu est : ", `tab[i+1]`) {7}
Fin

1. Corriger ces deux lexiques et ces deux algorithmes
2. Une fois les algorithmes corrigés :

(a) pour le premier algorithme :
— la taille du tableau sera fixée à 5 éléments
Indiquer quels sont les deux messages affichés à la fin de l'algorithme1.

(b) Pour le deuxième algorithme :
— on suppose que l'utilisateur entre `val`=7.4 et `i`=3.
Quel est le message affiché à la fin de l'algorithme2 ?
— Proposer les instructions permettant de copier la valeur modifiée de `tab` dans `tab2` (à la même position) puis d'afficher les valeurs renseignées dans le tableau `tab2`.

Voir la correction : 10.13

10.2 [Ch] Saisie d'un nombre inconnu de valeurs dans un tableau

Un utilisateur veut saisir et mémoriser une liste de nombres entiers naturels et indiquer qu'il a terminé en saisissant la valeur -1.

1. Proposer l'algorithme de cette fonction `Saisie_entiers` qui retournera le nombre d'éléments saisis.
2. Proposer ensuite l'algorithme d'une fonction dont le rôle est de retourner le nombre d'éléments nuls.
3. Proposer un algorithme principal permettant de tester ces fonctions et d'afficher en particulier le nombre d'éléments saisis, le nombre d'éléments nuls et le nombre d'éléments non nuls saisis par l'utilisateur.

Voir la correction : 10.14

10.3 [TD] Utilisation directe de tableaux

1. Déclarer un tableau de 7 réels dont seules les 4 premières cases sont initialisées avec les valeurs de votre choix.
2. Proposer une instruction permettant de calculer la moyenne de ces 4 éléments et de l'affecter à la 5^e case, puis afficher cette valeur.
3. Proposer une instruction permettant de calculer la moyenne des carrés de ces 4 éléments et de l'affecter à la 6^e case, puis afficher cette valeur.
4. Proposer une instruction permettant de calculer la différence entre la valeur contenue dans la 6^e case et le carré de la 5^e case et d'affecter le résultat à la 7^e case, puis afficher ces valeurs (resp. la moyenne et la variance des 4 premiers éléments).

10.4 [TD] Avec des Si (*)

1. Proposer le prototype et l'algorithme d'une fonction `Saisie_ordon` dont le rôle est de permettre à un utilisateur de remplir un tableau de 3 réels en les rangeant automatiquement dans l'ordre croissant après chaque saisie.
Plus précisément la fonction doit permettre :

— à un utilisateur de saisir une valeur en première position dans le tableau.
— à ce même utilisateur de saisir une deuxième valeur et de la ranger soit dans la deuxième case si elle est strictement plus grande que la première, soit la première valeur doit être déplacée dans la deuxième case et la seconde valeur dans la première case.
— etc...

2. Proposer un algorithme principal permettant de tester votre fonction.

10.5 [TD] Manipulation de tableaux et boucles (*)

Pour chaque cas, proposer l'algorithme (R, E, E/S, S, déclaration et définition) d'une fonction dont :

1. le rôle est de copier les élements d'un tableau d'entiers dans un autre.
2. le rôle est d'afficher les éléments d'un tableau d'entiers sous la forme : `{e1,e2,e3,...,en}`
3. le rôle est de tester si deux tableaux sont identiques
4. le rôle est de calculer la moyenne des éléments d'un tableau
5. le rôle est de remplacer les valeurs d'un tableau par sa somme cumulée :

$$T[n] = \sum_{k=0}^{n} T[k]$$

6. Proposer un algorithme principal dans lequel vous pourrez tester ces fonctions sur un tableau d'entiers, de taille suffisamment grande, qui sera rempli à l'aide de la fonction `Saisie_entiers` de l'exercice 10.2.

10.6 [TD] Tableaux à deux dimensions et boucles (**)

1. Déclarer un tableau à deux dimensions permettant de représenter la table :

02	03	05	07	11	13
17	19	23	29	31	37
41	43	47	53	59	61
67	71	73	79	83	89

2. Proposer l'algorithme d'une fonction dont le rôle est de permuter deux lignes dans le tableau.
3. Proposer l'algorithme d'une fonction dont le rôle est de permuter deux colonnes dans le tableau.
4. Proposer une fonction permettant d'afficher les éléments d'un tableau à deux dimensions sous la forme.

```
| e11 e12 ... e1n |
| e21 e22 ... e2n |
| ... ... ... ... |
| em1 em2 ... emn |
```

5. Proposer enfin un algorithme principal dans lequel vous testerez vos fonctions pour permuter la 1ère et la 3ème ligne ainsi que la 2ème et la 5ème colonne, puis vous afficherez le tableau après permutation.

10.7 [TD] Tableaux à deux dimensions (**)

Un carré magique est un carré de 3 cases par 3 cases contenant des nombres tels que la somme des nombres sur les lignes, les colonnes et les deux diagonales du carré sont identiques.

— Proposer le prototype et l'algorithme d'une fonction `Est_magique` permettant de dire (VRAI ou FAUX) si un carré passé en paramètre est magique.

— Proposer un algorithme principal permettant de tester votre fonction.

10.8 [TD] Interpolation linéaire dans un tableau (**)

Soit f un signal à valeurs réelles et n un entier naturel fixé une fois pour toute uniquement connu par ses valeurs f_k, $k \in \mathbb{N} \cap [0, n]$, échantillonnées aux abscisses réelles x_k espacées régulièrement avec un pas Δ réel, comme illustré sur la figure 10.1.

L'interpolation linéaire est très utilisée en traitement du signal pour évaluer le signal f en tout point $x \in [x_0, x_n]$. La formule de l'interpolation linéaire entre 2 points a été donnée dans l'exercice 2.10.

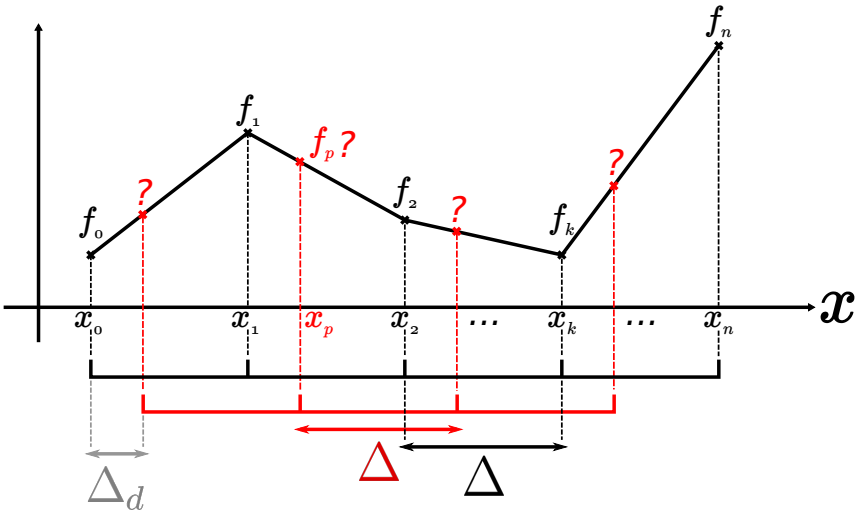


FIGURE 10.1 – Interpolation linéaire d'un signal f échantillonné régulièrement

Du point de vue algorithmique, les échantillons $\{f_k\}$ du signal et leurs abscisses $\{x_k\}$ sont stockés dans 2 tableaux respectifs de taille $n + 1$. Couramment, l'interpolation est réalisée sur un tableau d'abscisses $x_p \in [0, x_n]$, $p \in \mathbb{N} \cap [0, n-1]$, régulièrement espacées du pas Δ (le même que pour les x_k), et décalées d'un pas $\Delta_d < \Delta$ réel constant (cf. Fig. 10.1).

1. Définir une fonction dont le rôle est de remplir un tableau de taille $n+1$ contenant des valeurs d'abscisses x_k espacées régulièrement d'un pas Δ à partir d'une valeur x_0 initiale (cf. Fig. 10.1).
2. Proposer l'algorithme d'une fonction qui réalise l'interpolation linéaire du tableau d'échantillons $\{f_k\}$ aux points d'abscisses $\{x_k\}$ sur le tableau d'échantillons $\{f_p\}$ décalés aux points d'abscisses $\{x_p\}$. La fonction devra vérifier que les abscisses x_p sont bien toutes comprises entre x_0 et x_n.

3. Proposer un algorithme principal permettant de tester vos fonctions.
4. **Pour aller plus loin (***)** : modifier votre fonction d'interpolation linéaire pour qu'elle gère aussi les points x_p en dehors de l'intervalle $[0, x_n]$, en supposant que le signal f y est nul, i.e. $f_k = 0$ pour $x_k \in]-\infty, 0[\cup]x_n, \infty[$ et $k \in \mathbb{Z}$.

10.9 [TD] Chaînes de caractères et tableaux (*)

1. Proposer le prototype et l'algorithme d'une fonction `nomprenom` permettant de générer un tableau de caractères contenant les 3 premières lettres d'un prénom, suivies d'un tiret, suivi des 3 premières lettres d'un nom. Le nom et le prénom seront passés en paramètres de la fonction.
   Exemple : si on passe `Florent` et `Bernard` comme paramètres à la fonction, elle doit construire un tableau contenant : les caractères `'F','l','o','-','B','e','r','\0'`
   Votre algorithme doit également gérer les cas particuliers comme par exemple :
   (a) Zo Andriamanoharisoa (CiTiSE1 2013-2014)
   (b) Omar Sy
2. Proposer un algorithme principal permettant de tester votre fonction.

10.10 [TD] Chaînes de caractères (*)

Dans cet exercice, les chaînes seront toutes vues comme étant des tableaux de caractères.

1. Proposer l'algorithme d'une fonction `longueur_chaine` dont le rôle est de compter la longueur (nombre de caractères) d'une chaîne de caractères passée en paramètre.
2. Proposer l'algorithme d'une fonction dont le rôle est de compter le nombre de voyelles d'une chaîne de caractères passée en paramètre.
3. Proposer l'algorithme d'une fonction dont le rôle est de compter le nombre de mots dans une chaîne de caractères (on suppose que les mots sont uniquement séparés par un et un seul espace).
4. Proposer un algorithme principal permettant d'appeler ces fonctions.

10.11 [TD] Conversions nombre arbitrairement grand en tableau d'entiers (****)

Pour les plus courageux seulement. Exercice assez difficile [1].
Pour saisir un grand nombre, l'utilisateur le saisit en base 10. Cette information est considérée comme étant une chaîne de caractères.
Pour utiliser ce nombre dans un système informatique, il faut le convertir en un ensemble de mots machines (généralement sur 32 bits). Cela revient à connaître chacun de ses "chiffres" en base $r = 2^{32}$.
On souhaite donc écrire une fonction qui permet de convertir cette chaîne de caractères

1. en lien avec le projet Info2

représentant un grand entier en un tableau de chiffres de 32 bits.
Exemple :
Si l'utilisateur saisit la chaîne constituée des chiffres `''12345678910111213141516l7''` en base 10, votre fonction devra remplir un tableau constitué de :
`{3197516993, 255446423, 66926}` où les mots de 32 bits de poids faibles sont à gauche et les mots de 32 bits de poids fort sont à droite et devra retourner le nombre de chiffres en base $r = 2^{32}$, ici 3 correspondant à la taille pratique du tableau.
Cela signifie que :

$$66926 \times (2^{32})^2 + 255446423 \times 2^{32} + 3197516993 = 12345678910111213141516l7$$

Proposer l'algorithme d'une telle fonction, en précisant R, E, E/S, S.

10.12 [TD] Conversions nombre sous forme de tableau d'entiers en chaîne de caractères (****)

Proposer la fonction réciproque de l'exercice 10.11.

10.13 [Corr] A partir d'algorithmes donnés

1. Corriger ces deux lexiques et ces deux algorithmes

   Lexique1 :
   {Lors de la déclaration d'un tableau, on doit lui attribuer une taille maximale **fixée une fois pour toute**.
   Cette taille est donc une constante qui doit être déclarée et initialisée dans le Lexique} ~~`taille` : 1 entier~~ TAILLE_MAX : la constante entier :=5
   `tab` : 1 tableau de ~~`taille`~~ TAILLE_MAX réel
   {Au final, un tableau de taille 5 est déclaré. Chaque case du tableau contient un réel.
   Si on souhaite modifier la taille du tableau, on pourra modifier, à un seul endroit du code, l'initialisation de la constante TAILLE_MAX.}

   Algorithme1 :
   Début
   `tab[0]`⟵3.4 {1}
   `tab[`~~`taille`~~ `TAILLE_MAX-1]`⟵`tab[0]`-1 {2}
   Ecrire("Le premier élément de ce tableau est : ",~~`tab[1]`~~ `tab[0]`) {3}
   Ecrire("Le dernier élément de ce tableau est : ",~~`tab[taille]`~~`tab[TAILLE_MAX-1]`) {4}
   Fin
   Attention aux erreurs classiques ici. Les éléments du tableau sont numérotés de 0 à `TAILLE_MAX-1`.
   Le premier élément du tableau est celui d'indice 0, tandis que le dernier est celui d'indice `TAILLE_MAX-1`.
   Le 4ème élément du tableau est celui d'indice 3 et de manière plus générale : le n ième élément du tableau est celui d'indice n-1.

{ Ici la taille est inscrite une fois pour toute "en dur" pour les deux tableaux à travers l'utilisation de la constante entière 6.
Puis les deux tableaux sont initialisés.
Dans le premier cas, il y a plus de valeurs que la taille du tableau. Même si c'est une erreur de bon sens, ce n'est pas une erreur au niveau programmation.
Les deux valeurs supplémentaires sont perdues (4.3, -2.5) et les 6 premières sont bien mémorisées dans le tableau `tab`. On évitera ce genre d'ambiguïté qui ne provoque pas d'erreur de compilation mais qui n'apporte que de la confusion.
Dans le second cas, il y a moins de valeurs que la taille du tableau. Dans ce cas, seules les 3 premières cases sont initialisées et trois dernières restent non initialisées.
Une instruction du type Ecrire(`tab2[3]`) provoquera une erreur comme quoi vous essayez d'utiliser une variable qui n'a pas été initialisée (`The variable tab2[3] is being used without being initialized`)
En revanche l'utilisateur est libre d'entrer des données dans les cases "libres" de `tab2` et de les afficher ou d'effectuer des traitements dessus par la suite. }

Lexique2 :

`tab` : 1 tableau de 6 réel :={-4.2,5.4,8.4,9.3,-4.5,0}~~,4.3,-2.5}~~

`tab2` : 1 tableau de 6 réel :={-3.4,5.2,1.9}

`i` : un ~~réel~~ entier {C'est une erreur par rapport à l'utilisation qui est faite par la suite dans l'algorithme. `i` est l'indice qui permet de repérer les éléments du tableau, cet indice est nécessairement un entier.}

Algorithme2 :

Début

Ecrire("Saisir un ~~indice~~ entier pour voir quel est le i^e élément du tableau") {1}

{Ici, il y a confusion entre le terme indice et le terme ième élément. On demande bien à l'utilisateur d'entrer un entier (et pas un indice) correspondant au ième élément, c'est à dire l'élément d'indice i-1.}

Lire(`i`) {2}

Ecrire("Le i^e élément de ce tableau est : ",`tab[i-1]`) {3}

Ecrire("Quel est l'indice de la case que vous souhaitez modifier et quelle valeur souhaitez vous mettre ?") {4}

{Ici, on demande bien l'indice (entre 0 et la taille du tableau moins 1). On doit donc entrer la valeur à la position d'indice i (et pas i+1). On a en revanche modifié la (i+1) ème case du tableau !}

Lire(`i`,`val`) {5}

`tab[i~~+1~~]`⟵`val` {6}

Ecrire("Vous avez modifié la, ",i+1,"^{e} case. Son nouveau contenu est : ", `tab[i~~+1~~]`) {7}

Fin

2. Une fois les algorithmes corrigés :

   (a) pour le premier algorithme :

   — la taille du tableau sera fixée à 5 éléments

   Indiquer quels sont les deux messages affichés à la fin de l'algorithme1.

   A la fin de l'algorithme1, les deux messages affichés sont :

   `Le premier élément de ce tableau est: 3.4`

```
Le dernier élément de ce tableau est: 2.4
```

(b) Pour le deuxième algorithme :

— on suppose que l'utilisateur entre `val`=7.4 et `i`=3.
Quel est le message affiché à la fin de l'algorithme2 ?
A la fin de l'algorithme2, le message affiché est :
`Vous avez modifié la 4`^{e} `case. Son nouveau contenu est: 7.4`

— Proposer les instructions permettant de copier la valeur modifiée de `tab` dans `tab2` (à la même position) puis d'afficher les valeurs renseignées dans le tableau `tab2`. {Suite de l'algorithme2
Il y a, au final, 4 valeurs à afficher dans `tab2`}

```
tab2[i]⟵tab[i]                                              {8}
Ecrire("{",tab2[0],",",tab2[1],",",tab2[2],",",tab2[3],"}")  {9}
```

Retour à l'énoncé : 10.1

10.14 [Corr] Saisie d'un nombre inconnu de valeurs dans un tableau

Un utilisateur veut saisir et mémoriser une liste de nombres entiers naturels et indiquer qu'il a terminé en saisissant la valeur -1.

1. Proposer l'algorithme de cette fonction `Saisie_entiers` qui retournera le nombre d'éléments saisis.
2. Proposer ensuite l'algorithme d'une fonction dont le rôle est de retourner le nombre d'éléments nuls.
3. Proposer un algorithme principal permettant de tester ces fonctions et d'afficher en particulier le nombre d'éléments saisis, le nombre d'éléments nuls et le nombre d'éléments non nuls saisis par l'utilisateur.

{Exercice extrêmement important, mêlant boucles, parcours de tableaux et condition d'arrêt. A étudier et à bien comprendre}
{C'est une proposition de correction, il y a plusieurs façon de le réaliser, mais attention à bien tester !}
{Vous pouvez tester de la manière suivante :}
{On entre directement -1 : nombre d'élément saisi doit être à 0}
{On entre -1 dans le tableau : on doit retourner le nombre d'éléments saisis sans prendre en compte -1}
{On entre jamais -1 et on remplit tout le tableau, la fonction ne doit pas nous permettre d'écrire dans une case du tableau qui n'existe pas et nous sortir de la boucle dès que le tableau est complètement rempli, dans ce cas, la taille pratique sera la même que la taille réelle.}

{Définitions des fonctions}
{
R : Demande à l'utilisateur de saisir des éléments dans un tableau et retourne le nombre

d'éléments saisis. Important : on ne sait pas combien de valeurs l'utilisateur va entrer au départ.
E : 1 entier correspondant au nombre de valeur max à saisir
E/S : 1 tableau d'entiers {La fonction va modifier le tableau passé en paramètre}
S : Un entier correspondant au nombre d'éléments saisis
}
{Remarque **TRES** importante : on ne précise pas ici la taille **réelle** du tableau. C'est à l'algorithme principal (ou plus généralement la fonction appelante) de s'assurer que le tableau passé à la fonction est bien dimensionné.
En revanche, on précise à la fonction une valeur maximale de nombre de saisie pour ne pas que l'utilisateur écrive dans une case qui n'existe pas.}

Saisie_entiers : la fonction(n : 1 entier, tab : 1 tableau d'entiers)$\longrightarrow$ 1 entier
  Lexique {local}
    i : 1 entier

  Algorithme {local}
  Début
    {conditions d'initialisation}
    $i \leftarrow 0$ {on se place sur la première case du tableau}
    {Puis on demande à l'utilisateur d'entrer une première valeur}
    Ecrire("Entrer une", i+1, "eme valeur (-1 pour arrêter la saisie)")
    Lire(tab[i]) {on la stocke directement dans la 1ere case du tableau}
    Tant que tab[i]$\neq -1$ ET i<n-1
      $i \leftarrow i + 1$ {on se positionne sur la case suivante}
      Ecrire("Entrer une", i+1, "eme valeur (-1 pour arrêter la saisie)")
      Lire(tab[i]) {on la stocke directement dans la 1ere case du tableau}
    FinTantque
    {On sort de la boucle quand i vaut $n-1$ (dernière case atteinte) ou tab[i]=-1}
    {i désigne alors le nombre d'éléments saisis, sans compter -1}
    {Attention : -1 peut très bien n'avoir jamais été saisi, dans ce cas on est sorti de la boucle car on a rempli tout le tableau (i=n-1)}
    {Le nombre d'éléments saisis dans ce cas particulier est donc i+1=n.}
    {Traitement du cas particulier}
    Si tab[i]≠-1
      Alors
        i←i+1
    FinSi
    Retourner i
  Fin
{Pourquoi la condition $i < n-1$ est-elle présente ? Pourquoi la condition $i < n$ n'est pas possible dans cet exemple ?}
{Aurait-on pu faire cet algorithme avec une boucle Faire...Tant que ? si oui Comment ?}
{A quoi sert le dernier Si final ? Pourrait-on s'en passer ? Pourquoi ?}
{Est ce que dans le Si, on aurait pu remplacer l'instruction i←i+1 par i←n}
{Pourquoi retourne-t-on i à la fin ?}

```
{
R : Compte et retourne le nombre d'éléments nuls dans un tableau passé en paramètre.
E : tab un tableau d'entiers, t un entier correspondant à la taille pratique du tableau
E/S : vide {La fonction n'est pas censée modifier le tableau}
S : Un entier correspondant au nombre d'éléments nuls dans le tableau
}
Zero_in_tab : la fonction(tab : un tableau d'entiers, t : 1 entier)⟶ 1 entier
  Lexique {local}
   i : 1 entier
   z : 1 entier

  Algorithme {local}
  Début
   {conditions d'initialisation}
   i ← 0 {on se place sur la première case du tableau}
   z ← 0 {le nb d'éléments nuls vaut 0 au départ}
   Tant que i < t {Il n'est pas nécessaire de tester si tab[i]≠ −1 car on connait ici la taille
pratique du tableau}
     Si tab[i]=0 {on teste si l'élément courant est nul}
      Alors {Si c'est le cas}
        z ← z + 1 {on incrémente le compteur d'éléments nuls}
      {Sinon on ne fait rien, la valeur de z reste inchangée}
     Fin Si
     i ← i + 1
   FinTantque
   Retourner z
  Fin
{Pourquoi on a la condition i < t et pas i < n ? Cette instruction est-elle correcte ? Ne
devrait-on pas mettre i < (t − 1) ?

Lexique {principal}
{Déclaration des fonctions}
Saisie_entiers : la fonction(n : 1 entier, tab : 1 tableau d'entiers)⟶ 1 entier
Zero_in_tab : la fonction(tab : un tableau d'entiers, t : 1 entier)⟶ 1 entier

zeros : 1 entier
entiers_memo : 1 tableau de 100 entiers
taille_pratique : 1 entier

    Algorithme {principal}
Début
  taille_pratique ← Saisie_entiers(100, entiers_memo)
  zeros ←Zero_in_tab(entiers_memo, taille_pratique)
  Ecrire("Il y a dans votre tableau :")
  Ecrire(zeros," éléments nuls et ", taille_pratique−zeros," éléments non-nuls")
Fin
```

Retour à l'énoncé : 10.2

Chapitre 11

Les tableaux en C

11.1 [TD] Fonctions de manipulation de tableaux

L'objectif de cet exercice est de créer un fichier `tableaux.h` et un fichier de définitions associées `tableaux.cpp` regroupant les fonctions de base permettant de manipuler les tableaux, dont vous avez étudié les algorithmes dans les exercices du chapitre précédent.
Ces fichiers seront à conserver précieusement puisque vous pourrez vous en servir dès que vous aurez besoin de manipuler des tableaux dans d'autres contextes d'utilisation.

Parmi ces fonctions, vous devez impérativement déclarer, définir et **tester** :

1. Fonction de saisie d'éléments (de type réels) dans un tableau
2. Fonction d'affichage des éléments d'un tableau
3. Fonction de copie d'un tableau dans un autre
4. Fonction de comparaison entre deux tableaux
5. Libre à vous d'en ajouter d'autres par la suite et de compléter votre "bibliothèque" de fonctions manipulant les tableaux.

11.2 [TD] Tableaux et chaînes de caractères

Traduire en C l'algorithme de la fonction `nomprenom` de l'exercice 10.9 ainsi que son appel dans un algorithme principal.
Vous devez tester les cas particuliers.

11.3 [TD] Traduction des algorithmes du chapitre 10

Traduire les algorithmes étudiés dans le chapitre 10 des fonctions restantes.

Chapitre 12

Le type composé

12.1 [Ch] A partir d'un algorithme donné

Lexique :

`z` : un type composé de
<
`re` : 1 réel
`im` : 1 réel
>

Algorithme :

Début
z⟵3 {1}
z⟵3+4i {2}
z•re⟵3+4i {3}
z•re⟵3 {4}
z•im⟵4i {5}
z•im⟵z•re {6}
Fin

1. Corriger le lexique
2. Parmis les instructions précédentes (1-6), indiquer et justifier celles qui sont valides.
3. A la fin de l'algorithme,en supposant que seules les instructions valides ont été exécutées, quel est le contenu de la variable z (ie : quel est la valeur de chacun de ses champs) ?

voir la correction : 12.9

12.2 [Ch] Points

1. Déclarer un type point2d (2 coordonnées).
2. Proposer un découpage fonctionnel simple permettant à un utilisateur de saisir deux points et d'afficher leur distance

voir la correction : 12.10

12.3 [Ch] Menus

Dans les deux exercices précédents, on a utilisé deux champs de même type. Un des intérêts du type composé est de pouvoir représenter un "objet" qui a des caractéristiques de type différent.

Un restaurant propose des menus à la carte. Le patron du restaurant souhaite que le client puisse voir le menu avec 3 entrées au choix, le plat principal avec 4 choix possibles (deux viandes et deux poissons) et deux desserts au choix. Le prix de chaque plat doit être indiqué sur le menu.
Chaque plat est numéroté de la façon suivante :

- x pour les entrées où x désigne le numéro de l'entrée (entre 1 et 3)
- $1x$ pour les plats où x désigne le numéro du plat principal (entre 1 et 4)
- $2x$ pour les desserts où x désigne le numéro du dessert (entre 1 et 2)

Sur le menu, les plats sont affichés avec leur nom. A vous de trouver un moyen de faire correspondre le numéro du plat avec son nom.

1. Déclarer un nouveau type `choix` composé d'un entier (correspondant au numéro du plat) et d'un réel correspondant au prix du plat ;
2. Proposer un découpage fonctionnel simple permettant :
   - d'afficher le menu avec tous les choix possibles
   - de permettre au client d'entrer son choix pour chaque partie du repas (entrée, plat, dessert)
   - de lui afficher le prix total du menu une fois son choix effectué.

voir la correction : 12.11

[TD] Saisie d'un coureur (*) d'après un sujet d'examen final (2013-2014)

1. Un coureur est caractérisé par :
   - son nom : p.ex. U. Bolt
   - son dossard : p.ex. 2163
   - la distance de la course (en m) : p.ex. 100
   - son temps de référence (en s) : p.ex. 9.58
   - le temps réalisé pour la course.

   Déclarer, un type composé `coureur` permettant de représenter un coureur.

2. Déclarer une fonction `Saisie_coureur` dont le rôle est de demander à l'utilisateur de saisir les informations sur le coureur et qui retourne un `coureur`.

3. Déclarer une fonction `Affiche_coureur` qui permet d'afficher les caractéristiques d'un coureur.
4. Proposer l'algorithme principal permettant de saisir un coureur et d'afficher ses caractéristiques.
5. Définir les fonctions `Saisie_coureur` et `Affiche_coureur`.

12.4 [TD] Points et vecteurs (*)

1. Déclarer un type point3d et un type vecteur3d
2. Déclarer une constante ORIGINE de type point3d de coordonnées (0,0,0)
3. Proposer un découpage fonctionnel simple permettant à un utilisateur de saisir deux point3d M et N (par leurs coordonnées) et qui génère les vecteurs $\overrightarrow{OM}$ (où O est l'origine), $\overrightarrow{ON}$ et $\overrightarrow{MN}$.
4. Proposer deux nouvelles fonctions permettant
   — de calculer la norme de chacun de ces vecteurs ;
   — de déterminer (VRAI ou FAUX) si deux vecteurs sont perpendiculaires [1]

12.5 [TD] Complexes (*)

1. Déclarer un type complexe
2. Proposer les algorithmes de chacune des fonctions suivantes permettant à partir d'un complexe `z` de calculer :
   (a) son conjugué
   (b) son inverse
   (c) son opposé
   (d) son module
   (e) son argument
3. Proposer ensuite une fonction permettant d'afficher un `complexe`.
   Exemple : pour afficher le complexe `3+-4i`, on aimerait voir apparaitre `3-4i`.

12.6 [TD] type composé, tableaux et boucles (***)

1. Proposer un découpage fonctionnel simple permettant de :
   — Demander à un utilisateur d'entrer un certain nombre de `point2d` (Le type composé `point2d` est à créer) et de les mémoriser dans un tableau.
   — Puis d'afficher les coordonnées des deux points les plus proches au sens de la distance euclidienne.
   Rappel : La distance euclidienne entre deux points de coordonnées (x_A, y_A) et (x_B, y_B) est donnée par $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$
2. Proposer un algorithme principal permettant de résoudre le problème.

12.7 [TD] Synthèse (d'après sujet examen 2014-2015)

Un plateau de jeu est constitué d'un quadrillage dont les lignes sont repérées par des chiffres (entre 1 et 8) et les colonnes par des lettres (entre 'a' et 'h'). Chaque case peut contenir une pièce dont on donnera le nom limité à 20 caractères imprimables (représenté par un tableau de caractères modélisant une chaîne de caractères).

1. On rappelle que deux vecteurs sont perpendiculaires lorsque leur produit scalaire est nul

Lorsque une case ne contient pas de pièce, on donnera un nom correspondant à la chaîne de caractère `""`.

1. Déclarer un nouveau type `coord` permettant de représenter une case du jeu.
2. Proposer l'algorithme d'une fonction `case_valide` dont le rôle est de vérifier si les coordonnées d'une case sont valides. Cette fonction doit retourner VRAI si la case est une case du plateau et FAUX sinon.
   Exemple : si la case est de coordonnées f4, la fonction doit répondre VRAI. Si la case est de coordonnées a0 ou encore i3 la fonction doit répondre FAUX.
3. Proposer l'algorithme d'une fonction `Saisie_case` qui permet à un utilisateur de saisir et de **retourner une** case **existante** sur le plateau. Cette fonction doit donc impérativement tester la case entrée par l'utilisateur et redemander une saisie tant que celle ci n'est pas valide. Une fois que la case est existante, la fonction doit demander et mémoriser le nom de la pièce à placer sur cette case.
4. Proposer l'algorithme d'une fonction `Affiche_case` qui permet à un utilisateur d'afficher les caractéristiques d'une case du plateau.
5. Proposer l'algorithme d'une fonction `Saisie_emplacements` qui permet de **mémoriser** les positions de plusieurs pièces sur le plateau et qui **retourne** le nombre de positions mémorisées.
   La fonction proposera à l'utilisateur de saisir chacune des cases (grâce à la fonction `Saisie_case` qui doit impérativement être utilisée) et les mémorisera dans un tableau tant que l'utilisateur souhaite saisir une position. Un test doit impérativement être fait pour savoir si l'utilisateur souhaite s'arrêter ou pas. De plus, il n'y a que 64 cases sur le plateau, donc 64 positions maximales à mémoriser.
6. Proposer l'algorithme principal permettant de tester ces fonctions. L'algorithme principal devra afficher à l'utilisateur combien de positions il a saisi et afficher ensuite les caractéristiques de chacune des cases saisies.

12.8 [TD] Les 12 billes (****)

On dispose de 12 billes de même taille, de même couleur et indiscernables au toucher. Une et une seule de ces 12 billes n'a pas le même poids que les autres (elle peut être plus lourde, ou plus légère).
On dispose d'une balance Roberval (balance à plateaux) qui permet d'effectuer des comparaisons de deux masses disposées sur ses plateaux et indique quel plateau contient la masse la plus lourde.
L'objectif de l'exercice est de proposer un algorithme permettant de déterminer, en au plus 3 pesées, quelle est la bille n'ayant pas le même poids que les autres et d'indiquer si elle est plus lourde ou plus légère.

1. Dans un lexique principal :
   (a) Définissez le type composé `Bille` composé d'un entier représentant le numéro de la bille (entre 1 et 12) et d'un réel représentant la masse de la bille
   (b) L'ensemble des 12 billes sera représenté par un tableau de 12 `Bille`. Déclarez ce tableau.

2. Déclarez, en précisant R,E,S, la fonction `Ajoute_masses` qui à partir d'un tableau de `Bille` et d'un tableau de numéros de `Bille` ajoute les masses des billes dont les numéros sont dans le tableau de numéros.
3. Déclarez, en précisant R,E,S, la fonction `Pese_Billes` qui permet de comparer les masses de deux plateaux (A et B pour fixer les idées) de `Bille`. Cette fonction devra retourner -1 si le plateau A est plus lourd que le plateau B, 0 si le plateau A et le plateau B sont à l'équilibre et 1 si le plateau A est plus léger que le plateau B.
4. Déclarez, en précisant R,E,S, la fonction `Initialise_Billes` qui permet d'affecter une masse identique à 11 billes et une masse différente à une 12 bille. Le choix des masses et du numéro de la bille n'ayant pas le même poids que les 11 autres devra être fait au hasard [2].
5. Déclarez, en précisant R,E,S, la fonction `En3pesee` dont le rôle est de déterminer, en 3 pesées maximum, quel est le numéro de la bille qui n'a pas le même poids que les autres et d'indiquer si elle est plus lourde ou plus légère.
   Comme une fonction ne peut retourner qu'un seul type d'information, on pourra utiliser la convention suivante : la fonction retourne -x (où x est un entier entre 1 et 12) dans le cas où la bille est plus légère et +x dans le cas où la bille est plus lourde.
6. Proposez l'algorithme principal permettant de résoudre le problème. L'algorithme principal doit afficher à l'utilisateur le message : « La bille x est plus lourde que les autres » ou « La bille x est plus légère que les autres »
7. Donnez les définitions de chacune des fonctions du découpage.
   Remarque : La fonction `En3pesee` doit utiliser les fonctions `Ajoute_masses` et `Pese_Billes` et doit afficher à l'utilisateur à quelle pesée il en est.
8. Pour vérifier le résultat, vous pouvez proposer une fonction d'affichage du tableau de `Bille` dont le rôle est d'afficher chaque élément, c'est à dire le numéro de la bille et sa masse (qui ont été tirés au hasard via la fonction `Initialise_Billes`.

12.9 [Corr] A partir d'un algorithme donné

1. Corriger le lexique
   Une des plus grosses erreurs est de confondre la déclaration d'un nouveau type et la déclaration de la variable de ce type.
   Les deux étapes sont indispensables et doivent être effectuées dans l'ordre suivant :
   (a) On déclare le nouveau type.
   (b) On déclare une variable de ce type là.
   Lexique :

2. voir les fonctions `rand()` et `srand()`

{On déclare **d'abord** le nouveau type}
~~z~~ `complexe` : un type composé de

<
`re` : 1 réel
`im` : 1 réel
>

{**Puis** on déclare une variable de ce type là}
`z` : un `complexe`

Algorithme :
Début

~~z←3~~	{1}
~~z←3+4i~~	{2}
~~z•re←3+4i~~	{3}
z•re←3	{4}
~~z•im←4i~~	{5}
z•im←z•re	{6}

Fin

2. Parmis les instructions précédentes (1-6), indiquer et justifier celles qui sont valides.

   {1} : n'est pas valide car la partie gauche de la flèche d'affectation est une variable de type `complexe` alors que la partie droite est de type réel. Il y a violation du principe de cohérence des types.
   {2} : Il faut se poser la question de savoir ce que représente 3+4i. La réponse est simple, cette écriture n'a aucun sens d'un point de vu algorithmique. La confusion peut venir du fait que nous représentons les nombres complexes ainsi sous forme algébrique en mathématiques.
   En programmation, la machine n'a aucune idée de ce qu'est un complexe. Nous lui avons juste indiqué que c'est la composition de deux informations de type réel.
   Cette instruction est donc fausse car la partie droite de la flèche d'affectation n'a aucun sens.
   {3} : Pour la même raison qu'en {2} cette instruction est fausse.
   {4} : Cette instruction est correcte. la partie gauche est le champ `re` de la variable `z` qui est de type réel. La partie droite est le réel 3.
   {5} : Cette instruction est fausse. Qu'est ce que signifie 4i en algorithmie ? Quel doit être le type de la partie droite ? Comment dire que `z` a pour partie imaginaire 4 ?
   {6} : Ici l'instruction est correcte. `z•re` est évalué, est de type réel et retourne la valeur réelle 3.
   Cette valeur est affectée à `z•im` qui est aussi de type réel.

3. A la fin de l'algorithme, en supposant que seules les instructions valides ont été exécutées, quel est le contenu de la variable z (ie : quel est la valeur de chacun de ses champs) ?
   `z•re` contient la valeur 3 et `z•im` contient également la valeur 3.

`z` correspond donc au nombre complexe (en notation algébrique qui n'a de sens qu'en mathématique) 3+3i.

retour à l'énoncé : 12.1

12.10 [Corr] Points

1. Déclarer un type point2d (2 coordonnées).
2. Proposer un découpage fonctionnel simple permettant à un utilisateur de saisir deux points et d'afficher leur distance

Lexique principal :
{
R: Demande à l'utilisateur de saisir l'abscisse et l'ordonnée d'un point et retourne ce point
E: vide
S: 1 point2d
{**Remarque importante :** On retourne toujours **un seul** type même s'il est composé de deux informations}
}
Saisie_point : la fonction(vide)→ 1 point2d

{
R: Calcule et retourne la distance entre deux points passés en paramètre.
E: 2 point2d dont on souhaite calculer la distance
S: 1 réel correspondant à la distance entre ces deux points.
}
Distance : la fonction(p : 1 point2d, q : 1 point2d)→ 1 réel

{Pour l'affichage de la distance, nous utiliserons simplement la fonction Ecrire.}

{On déclare **d'abord** le nouveau type}
point2d : un type composé de
<
x : 1 réel
y : 1 réel
>
{Il ne faut pas oublier de déclarer les points sur lesquels on va travailler dans l'algorithme principal} p1, p2 : 2 point2d
dist : 1 réel

Algorithme {principal} :
Début
p1← Saisie_point()
p2← Saisie_point()
dist← Distance(p1,p2)
Ecrire(''La distance entre vos deux points est de '', dist,''unités'')
Fin

{Définitions des fonctions}
Saisie_point: la fonction()→ 1 point2d

```
    Lexique {local}:
     p: 1 point2d {Le type point2d a bien été créé dans le lexique principal
globalement.}

    Algorithme {local}:
    Début
     Ecrire(''Entrez l'abscisse et l'ordonnée de votre point'')
     Lire(p.x,p.y) {On peut renseigner directement l'abscisse et l'ordonnée
du point dans les champs x et y du point2d p}
     Retourner p
    Fin

Distance: la fonction(p: 1 point2d, q: 1 point2d)→ 1 réel
    Lexique {local}:
      d: 1 réel
    Algorithme {local}:
    Début
     d← √((p.x − q.x)² + (p.y − q.y)²)
     Retourner d
    Fin
```

retour à l'énoncé : 12.2

12.11 [Corr] Menus

1. Déclarer un nouveau type `choix` composé d'un entier (correspondant au numéro du plat) et d'un réel correspondant au prix du plat ;
   La déclaration se fait dans le Lexique principal.
   `choix` : 1 type composé de :
   <
   n_plat : 1 entier
   prix : 1 réel
   >

2. Proposer un découpage fonctionnel permettant :
   — d'afficher le menu avec tous les choix possibles
   — de permettre au client d'entrer son choix pour chaque partie du repas (entrée, plat, dessert)
   — de lui afficher le prix total du menu une fois son choix effectué.
   En plus des fonctions qui apparaissent clairement dans le découpage, il convient de proposer une fonction qui fait le lien entre le numéro du plat et le nom du plat et qui permet de donner son prix.
   Cela permet, en cas de changement de menu, de ne modifier que cette fonction.

   Lexique {principal}
   {
   R : Affiche le nom du plat correspondant au numéro passé en paramètre et retourne

son prix
E : 1 entier correspondant au numéro du plat
S : 1 réel correspondant au prix du plat affiché
}
`Correspondance_num_nom_plat` : la fonction(n : 1 entier)⟶ 1 réel

{
R : Affiche le Menu du restaurant au client
E : vide
S : vide
Remarque : cette fonction va appeler la fonction `Correspondance_num_nom_plat`
}
`Affiche_Menu` : la fonction(vide)⟶ vide

{
R : Demande au client de choisir un numéro d'entrée et retourne le `choix` fait.
E : vide
S : 1 `choix` correspondant à l'entrée sélectionnée.
}
`Saisie_entree` : la fonction(vide)⟶1 `choix`

{
R : Demande au client de choisir un numéro de plat principal et retourne le `choix` fait.
E : vide
S : 1 `choix` correspondant au plat sélectionné.
}
`Saisie_plat` : la fonction(vide)⟶1 `choix`

{
R : Demande au client de choisir un numéro de dessert et retourne le `choix` fait.
E : vide
S : 1 `choix` correspondant au dessert sélectionné.
}
`Saisie_dessert` : la fonction(vide)⟶1 `choix`

{ R : Calcule le prix total du repas.
E : 3 choix correspondant à l'entrée, au plat principal et au dessert
S : 1 réel correspondant au prix du repas.
} `Addition` : La fonction(ent : 1 choix, plat : 1 choix, dess : 1 choix)⟶ 1 réel

{Déclaration du type composé `choix`}
`choix` : 1 type composé de :
<

```
                    n_plat : 1 entier
                    prix : 1 réel
                >

entree, plat, dessert : 3 choix
montant_menu : 1 réel


Algo {principal}
Début
  Affiche_Menu()
  entree←Saisie_entree()
  plat←Saisie_plat()
  dessert←Saisie_dessert()
  montant_menu←Addition(entree, plat, dessert)
  Ecrire("Le coût total pour votre repas est : ", montant_menu," euros")
  Ecrire("Merci de votre visite.")
Fin

{Algos des fonctions du découpage fonctionnel}
Correspondance_num_nom_plat : la fonction(n : 1 entier)⟶ 1 réel
   Lexique {local}
     tarif : 1 réel
   Algo {local}
   Début
     Selon(n)
       Cas 1 : Ecrire("Salade du chef")
               tarif←3.95
       Cas 2 : Ecrire("Assiette de charcuterie")
               tarif←4.95
       Cas 3 : Ecrire("Velouté de légumes")
               tarif←2.95
       Cas 11 : Ecrire("Boeuf bourguignon et ses pâtes fraiches")
                tarif←8.95
       Cas 12 : Ecrire("Paupiettes de veau et son riz long")
                tarif←7.95
       Cas 13 : Ecrire("Dorade avec sauce et citron et boulettes de riz")
                tarif←8.95
       Cas 14 : Ecrire("Pavé de saumon grillé sur lit de lentilles du Puy")
                tarif←7.95
       Cas 21 : Ecrire("Fondants au chocolat")
                tarif←3.95
       Cas 22 : Ecrire("Surprise glacée")
                tarif←2.95
       autrement : Ecrire("Erreur de saisie")
                   tarif←-1
```

```
    FinSelon
    Retourner tarif
  Fin

Affiche_Menu : la fonction(vide) ⟶ vide
  Lexique {local}
    tarif : 1 réel
  Algo {local}
  Début
    Ecrire("Bienvenue au restaurant : à la bonne fourchette")
    Ecrire("                    ~~~~                    ")
    Ecrire("Au menu :")
    Ecrire("Entrées :")
    Ecrire("~~~~")
    Ecrire("01 : ")
    tarif←Correspondance_num_nom_plat(1)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("02 : ")
    tarif←Correspondance_num_nom_plat(2)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("03 : ")
    tarif←Correspondance_num_nom_plat(3)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("                    ~~~~                    ")
    Ecrire("Viandes :")
    Ecrire("~~~~")
    Ecrire("11 : ")
    tarif←Correspondance_num_nom_plat(11)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("12 : ")
    tarif←Correspondance_num_nom_plat(12)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("Poissons :")
    Ecrire("~~~~")
    Ecrire("13 : ")
    tarif←Correspondance_num_nom_plat(13)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("14 : ")
    tarif←Correspondance_num_nom_plat(14)
    Ecrire(" :                  ", tarif,"euros")
    Ecrire("                    ~~~~                    ")
    Ecrire("Desserts :")
    Ecrire("~~~~")
    Ecrire("21 : ")
    tarif←Correspondance_num_nom_plat(21)
```

```
    Ecrire(" :                    ", tarif,"euros")
    Ecrire("22 : ")
    tarif←Correspondance_num_nom_plat(22)
    Ecrire(" :                    ", tarif,"euros")
    Ecrire("                   ~~~~                    ")
    Ecrire("              Bon appétit !                 ")
  Fin
```

{Le point difficile dans cette fonction est de bien comprendre l'appel qui est fait à la fonction `Correspondance_num_nom_plat`. D'une part elle affiche le nom du plat correspondant au numéro passé en paramètre. D'autre part, elle retourne un prix pour ce plat. Ce prix est récupéré dans la variable `tarif` puis est affiché à la suite du nom du plat.
Cet affichage est précédé de l'affichage du numéro du plat pour chacune des entrées du menu.
}

`Saisie_entree` : la fonction(vide)⟶1 `choix`

```
  Lexique {local}
    ent : 1 choix

  Algo {local}
  Début
    Ecrire("Entrer le numéro de votre entrée")
    Lire(ent.n_plat) {Il est tout à fait possible de remplir directement le champ n_plat de ent}
    Ecrire("Vous avez sélectionné :")
    ent.prix←Correspondance_num_nom_plat(ent.n_plat)
{ On affiche le nom de l'entrée sélectionnée et on stocke son prix dans le champ prix de notre variable ent }
    Retourner ent
  Fin
```

`Saisie_plat` : la fonction(vide)⟶1 `choix`

```
  Lexique {local}
    plat : 1 choix

  Algo {local}
  Début
    Ecrire("Entrer le numéro de votre plat")
    Lire(plat.n_plat) {Il est tout à fait possible de remplir directement le champ n_platde plat}
    Ecrire("Vous avez sélectionné :")
    plat.prix←Correspondance_num_nom_plat(plat.n_plat)
{ On affiche le nom du plat sélectionné et on stocke son prix dans le champ prix de notre variable plat }
    Retourner plat
  Fin
```

`Saisie_dessert` : la fonction(vide)⟶1 `choix`

Lexique {local}
dess : 1 `choix`

Algo {local}
Début
Ecrire("Entrer le numéro de votre plat")
Lire(dess.n_plat) {Il est tout à fait possible de remplir directement le champ n_plat de dess}
Ecrire("Vous avez sélectionné :")
dess.prix←`Correspondance_num_nom_plat`(dess.n_plat)
{ On affiche le nom du dessert sélectionné et on stocke son prix dans le champ prix de notre variable dess }
Retourner dess
Fin

`Addition` : La fonction(ent : 1 choix, plat : 1 choix, dess : 1 choix)⟶ 1 réel
Lexique {local}
prix_repas : 1 réel

Algo {local}
Début
prix_repas⟵ ent.prix+plat.prix+dess.prix
Retourner prix_repas
Fin

Retour à l'enoncé : 12.3

Chapitre 13

Le type composé en C

13.1 [C] Ensemble de fonctions sur les complexes

L'objectif de cet exercice est de proposer un ensemble de fonctions permettant de travailler sur des complexes.

1. On vous demande donc, en respectant le découpage en fichiers `.cpp` et `.h`, de fournir une implantation en C des fonctions permettant de manipuler les complexes.
   Parmi celles-ci, on retrouve :
   (a) `saisir_complexe` (algo vu en cours)
   (b) `affiche_complexe` (algo vu en cours)
   (c) `conjugue` (algo vu en TD)
   (d) `oppose` (algo vu en TD)
   (e) `inverse` (algo vu en TD)
   (f) `module` (algo vu en TD)
   (g) `argument` (algo vu en TD)
2. On vous demande également de proposer des fonctions permettant d'effectuer les opérations de base sur les complexes
   (a) `add_complexe`
   (b) `mult_complexe`
   (c) `div_complexe`

   Chaque fonction devra être testée puis validée dans un programme principal.

13.2 Application

Utiliser les fonctions précédentes pour résoudre le problème de calcul du module de la fonction de transfert suivante :

$$H(p) = \frac{14(p+2)}{(p+5)(p^2+6p+25)}$$

1. Proposer une fonction permettant de calculer $H(p)$ pour p un complexe.

2. L'utilisateur de votre programme devra saisir une pulsation ω. Le programme calculera alors $R(\omega) = |H(j\omega)|$ et affichera sa valeur.
   Proposer le programme principal permettant de réaliser ceci.

13.3 Traduction des exercices du chapitre 12

Traduire en C les autres exercices du chapitre 12

Chapitre 14

Les pointeurs

[Ch] Exercice 1 : Manipulation de Pointeurs

Dans un programme principal :

1. Déclarer deux variables de type `float`, `x` et `y`
2. Déclarer un pointeur vers des réels `p_f`
3. En utilisant le pointeur, affecter la valeur `-2.3` à `x`
4. Vérifier que l'affectation à l'aide du pointeur a bien fonctionné en affichant à l'écran le contenu de la variable `x`
5. Afficher l'adresse contenue dans `p_f`, afficher ensuite l'adresse de `x` et celle de `y`
6. Procéder de même pour affecter à `y` la valeur `7.4` et vérifier en affichant directement le contenu de `y` que son contenu est bien `7.4`
7. Afficher l'adresse contenue dans `p_f`, afficher ensuite l'adresse de `x` et celle de `y`
8. Vérifier que la valeur de `x` est inchangée

[Ch] Exercice 2 : Pointeurs et tableau

Dans un programme principal :

1. Déclarer un tableau de 10 entiers
2. Déclarer un pointeur vers des entiers
3. Faire pointer le pointeur sur la première case du tableau
4. Quelle est l'adresse contenue dans votre pointeur ?
5. En utilisant le pointeur, ranger la valeur `1` dans la case pointée par votre pointeur
6. Que donne l'instruction `p_i=tab;` ? (Vous pourrez afficher le contenu de `p_i` après cette instruction).
7. Que représentent `p_i+1`, `p_i+2`, `p_i+3` ? (Vous pourrez afficher ces différentes variables). Quelle est la différence entre deux adresses consécutives ? Pourquoi ? Aurait-on la même chose pour des pointeurs de caractères (`char`) ? de réels (`double`) ?
8. En utilisant le pointeur, faire en sorte que votre tableau contienne les valeurs des carrés des 10 premiers entiers non nuls.

9. Afficher le 5e élément du tableau, en utilisant le pointeur, en utilisant la notation indicielle du tableau, en utilisant la notation indicielle du pointeur. Conclusion ?

Exercice 3 : Fonction d'echange du contenu de deux variables

Un étudiant propose une fonction dont le rôle est d'échanger le contenu de deux variables passées en paramètre.
Il écrit :

```
//Fichier fonctions.h
void Echangef(float x, float y);
```

```
//Fichier fonctions.cpp
void Echangef(float x, float y)
{
 float temp;
 temp=x;
 x=y;
 y=temp;
}
```

```
//Fichier main.cpp
#include<iostream>
using namespace std;

#include"fonctions.h"

int main()
{
 float a=2.8, b=-5.6;
 cout<<"Avant echange: a="<<a<<" et b= "<<b<<endl;
 Echangef(a,b);
 cout<<"Après echange: a="<<a<<" et b= "<<b<<endl;
 return 0;
}
```

1. Quel résultat va-t-il obtenir ? Pourquoi ?
2. Proposer le prototype et la définition d'une fonction `Echange` qui permet d'échanger le contenu de deux variables de type réel.
3. Proposer un programme de test.

Exercice 4 : Fonction manipulant des pointeurs

On donne le prototype suivant :

```
/*
R: Affiche le contenu d'une zone mémoire contenant nb cases contigües de type
réel1/350e6
E: L'adresse de la zone, le nombre de cases de la zone
```

```
E/S: vide
S: vide
*/
void Affiche_réels(const float* p_f, unsigned int nb);
```

1. Proposer la définition de cette fonction.
2. Proposer un programme de test.

Exercice 5 : Allocation dynamique

1. Dans un fichier `main.cpp` seul. Réaliser un programme principal permettant de :
   (a) demander à un utilisateur de saisir un nombre de réels à stocker ;
   (b) créer une zone de cette taille là et mémoriser son adresse dans un pointeur vers des réels ;
   (c) remplir la zone ainsi créée avec les valeurs de votre choix (vous pouvez faire une boucle de saisie) ;
   (d) afficher le contenu de cette zone en réutilisant la fonction de l'exercice 4 ;
   (e) supprimer la zone. Pourquoi est-ce indispensable ?
2. A l'aide de fonctions :
   (a) Proposer l'algorithme d'une fonction dont le rôle est de créer une zone permettant de stocker un nombre `nb` de réels passé en paramètre. Cette fonction devra retourner l'adresse de la zone créée.
   (b) Proposer un programme de test dans lequel vous créerez et afficherez **successivement** 3 zones distinctes en utilisant **le même** pointeur.