

# L1 Informatique

## Algorithmique 1 – Exercices<sup>1</sup>

### 1 Instructions de base

#### 1.1 Premier programme ①

Écrire un programme qui demande à l'utilisateur de saisir deux nombres, puis qui affiche le produit de ces nombres. L'affichage doit être présenté comme suit :

```
Entrez un entier : 12
Entrez un autre entier : 7
Le produit de 12 par 7 est 84.
```

#### 1.2 Compilation et exécution [TP]

Compiler et exécuter le programme de l'exercice 1.1.

→ Commande de compilation : `g++ -Wall fichier_code_source.cpp -o fichier_executable.out`

#### 1.3 Fonction mathématique ①

Afficher la valeur de  $f(x) = (x^2 - 3x + 0.5)(x - 3)$  pour  $x$  saisi par l'utilisateur au clavier.

#### 1.4 Moyenne [TP]

1. Écrire un programme qui demande à l'utilisateur les notes obtenues par un étudiant à 3 épreuves (Anglais, Mathématiques, Informatique) de coefficients respectifs 2, 5 et 3. Le programme calcule ensuite la moyenne. L'affichage obtenu à l'écran doit suivre celui de l'exemple suivant :

```
Note en anglais (coeff.2) : 11
Note en mathématiques (coeff.5) : 8
Note en informatique (coeff.3) : 18
Moyenne obtenue : 11.6
```

2. Si cela n'a pas été déjà fait, utiliser des constantes pour gérer les coefficients.

#### 1.5 Échange de valeurs

Écrire un programme dans lequel deux variables entières  $v1$  et  $v2$  sont déclarées, puis leurs valeurs saisies par l'utilisateur. Le programme devra ensuite échanger les valeurs de  $v1$  et  $v2$  ( $v1$  prend la valeur de  $v2$  et réciproquement), puis afficher leurs nouvelles valeurs.

#### 1.6 Permutation de valeurs [TP]

1. Écrire un programme effectuant la permutation de trois variables  $v1$ ,  $v2$  et  $v3$ , c'est-à-dire que  $v1$  prend la valeur de  $v2$ ,  $v2$  celle de  $v3$  et  $v3$  celle de  $v1$ .
2. Modifier le programme afin de permuter quatre variables.

<sup>1</sup>Légende : ① Application directe du cours. [TP] Exercice de TP à tester sur machine. ② Exercice supplémentaire (travail personnel). ★ Exercice ou question plus difficile. Dans les exemples d'exécution, les valeurs en gras indiquent des lectures au clavier.

## 2 Alternatives (instructions conditionnelles)

### 2.1 Valeur absolue et parité ⑨

Écrire un programme qui demande à l'utilisateur de saisir un entier et qui calcule et affiche à l'écran la valeur absolue de l'entier saisi. On affichera également si la valeur absolue est un nombre pair ou impair.

### 2.2 Saisie d'une note ⑨

Écrire un programme qui demande à un utilisateur d'entrer une note et vérifie si elle est correcte (comprise entre 0 et 20). Si la note est correcte, le message `Note correcte` sera affiché à l'écran, dans le cas contraire `Note incorrecte` sera affiché. Faire une version sans opérateur logique et une avec.

### 2.3 Fonction mathématique 2 [TP]

Afficher la valeur de  $f(x) = \frac{|x|+3x^2}{\frac{2}{3}x^2-6}$  pour  $x$  saisi par l'utilisateur. Faire attention aux points suivants :

- Si  $f(x)$  n'est pas définie en  $x$ , il faut l'indiquer et ne pas effectuer une division par 0.
- Pour intégrer la valeur de  $\frac{2}{3}$  au calcul, il faut s'assurer d'effectuer une division réelle et pas une division entière.

### 2.4 Racines réelles d'une équation du second degré

Écrire un programme permettant à l'utilisateur de saisir les coefficients d'une équation du second degré (du type  $ax^2 + bx + c = 0$ ). Le programme résoudra ensuite l'équation et affichera la (les) racine(s) dans  $\mathbb{R}$ .

### 2.5 Racines complexes d'une équation du second degré [TP]

Étendre le programme de l'exercice 2.4 afin de résoudre l'équation dans  $\mathbb{C}$ .

### 2.6 Premier tri

Écrire un programme demandant à l'utilisateur de saisir trois valeurs entières, stockées dans des variables `v1`, `v2` et `v3`. Le programme triera ensuite les valeurs des variables dans l'ordre croissant : `v1` devra contenir la plus petite valeur, `v2` celle située entre les deux autres et `v3` la plus grande. Pour vérifier, on affichera enfin les valeurs des trois variables.

### 2.7 Calcul mental [TP]

1. Écrire un programme éducatif qui demande à un premier utilisateur (parent ou enseignant) deux nombres entiers `a` et `b`, puis qui demande à un second utilisateur (élève) de saisir la somme de ces deux entiers. Le programme vérifie enfin si l'entier saisi par l'utilisateur est effectivement la somme de `a` et `b`. Si tel est le cas, le programme affichera `Bravo`, dans le cas contraire, le programme affichera `Erreur` avec la solution.
2. Compléter ce programme de manière à proposer à l'utilisateur de choisir parmi trois opérations (+, -, \*) avant de saisir les deux opérandes.

## 2.8 Dates

1. Écrire un programme permettant de tester si une année saisie au clavier est bissextile. Selon la règle du calendrier grégorien (instauré en 1582), une année est bissextile si elle est divisible par 4 mais pas divisible par 100, à moins qu'elle soit divisible par 400. Par exemple, les années 2003, 1995, 1900 et 800 ne sont pas bissextiles, alors que les années 2012, 2000 et 1600 le sont.
2. ⑤ En reprenant le programme précédent, écrire un programme qui détermine le nombre d'années séparant deux dates valides. Il s'agit de demander à l'utilisateur deux dates (sous la forme de trois entiers à chaque fois représentant le jour, le mois, et l'année), de tester si ces deux dates sont correctes (en tenant compte du nombre de jours des mois, ainsi que des années bissextiles) et bien ordonnées, et si oui d'afficher le nombre d'années pleines écoulées entre elles comme dans l'exemple ci-dessous :

```
Date 1 : 1 12 1081
Date 2 : 1 8 1137
Années séparant ces deux dates : 55
```

## 3 Boucles

### 3.1 Factorielle ②

Écrire un programme qui calcule la factorielle d'un entier  $n$ .

### 3.2 Table de multiplication [TP]

Écrire le programme qui demande un nombre  $x$  à l'utilisateur, puis affiche la table de multiplication associée.

### 3.3 Vérification de saisies ②

1. Écrire un programme qui lit un réel positif et vérifie sa validité. Son exécution doit être de la forme :

```
Donner un réel positif : -8.7
Valeur incorrecte, donner un réel positif : -4
Valeur incorrecte, donner un réel positif : 8.75
Merci
```

2. Même question, avec un affichage respectant la forme suivante :

```
Donner un réel positif : -8.7
Donner un réel positif : 11
Merci
```

### 3.4 Moyenne de notes [TP]

1. Écrire le programme de calcul de la moyenne d'une suite de notes en terminant la saisie par -1. À chaque saisie on vérifiera que la note est comprise dans l'intervalle 0 à 20.
2. Enrichir le programme afin d'afficher également la plus grande et la plus petite note saisie.

### 3.5 Affichage d'une ligne de symboles [TP]

1. Écrire un programme qui, après avoir demandé à l'utilisateur un nombre entier  $n$ , affiche  $n$  étoiles sur une même ligne. Par exemple, si  $n = 3$ , le programme affichera \*\*\*.
2. Modifier le programme pour qu'il affiche alternativement une étoile puis un point d'exclamation. Au total,  $n$  symboles doivent être affichés sur la ligne. Par exemple, si  $n = 7$ , le programme affichera \*!\*\*\*!\*

### 3.6 Affichage de formes [TP]

En utilisant les caractères '\*' et ' ' (espace), afficher à l'écran les formes suivantes (la longueur  $l$  du côté sera demandée à l'utilisateur) : carré plein (composé de  $l^2$  étoiles), carré vide (des étoiles uniquement sur les côtés), damier (alternance des caractères sur les lignes et les colonnes), et triangle rectangle isocèle (quatre orientations possibles).

Exemple de formes à afficher pour une longueur  $l = 6$  :

*****	*****	* * *	*	*****	*****	*
*****	*   *	* * *	**	*****	*****	**
*****	*   *	* * *	***	****	****	***
*****	*   *	* * *	****	***	***	****
*****	*   *	* * *	*****	**	**	*****
*****	*****	* * *	*****	*	*	*****

### 3.7 Liste de nombres [TP]

Afficher (dans l'ordre croissant) tous les nombres de 100 à 999 ayant trois chiffres distincts dont la somme vaut 9.

### 3.8 Maximum

On souhaite rechercher dans une liste de nombres entiers positifs entrés au clavier la plus grande valeur saisie. Envisager différentes manières pour demander la saisie de plusieurs valeurs.

### 3.9 Nombres premiers

1. Écrire un programme qui affiche tous les diviseurs d'un nombre entier strictement positif saisi par l'utilisateur. On affichera également le nombre de diviseurs du nombre.
2. En déduire un programme déterminant si un nombre est premier ou non. Proposer des modifications permettant de rendre l'algorithme plus efficace.
3. On souhaite maintenant demander à l'utilisateur de saisir successivement plusieurs nombres premiers. Modifier le programme en conséquence, de manière à ce qu'il demande des nombres à l'utilisateur et qu'il s'arrête dès qu'un nombre non premier est saisi.

### 3.10 Nombre parfait [TP]

Un nombre entier naturel  $n$  est dit parfait s'il est égal à la moitié de la somme de ses diviseurs (1 et  $n$  inclus). On peut vérifier que les quatre premiers nombres parfaits sont 6, 28, 496 et 8128.

1. Écrire un programme qui détermine si un nombre donné est parfait.
2. Écrire un programme qui affiche tous les nombres parfaits inférieurs à une borne donnée.

### 3.11 Suites numériques

Écrire un programme permettant d'afficher les  $n$  premières valeurs des suites numériques suivantes :

1. 0 -2 4 -6 8 -10 12 -14 ...
2. 0 1 3 6 10 15 21 28 ...
3. 1 2 4 8 16 32 64 128 ...

4. 1 2 3 4 5 1 2 3 4 5 1 2 ...

5. 1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 1 2 ...

### 3.12 Nombre inconnu [TP] <sup>2</sup>

1. Écrire un programme permettant de jouer au nombre inconnu. Un nombre entier aléatoire entre 1 et 100 est stocké en mémoire et l'utilisateur fait des propositions de nombres. À chaque fois, on lui indique si le nombre qu'il a proposé est trop petit ou trop grand. Lorsque le nombre inconnu est trouvé, on affichera le nombre de propositions faites.
2. ★ Les rôles sont inversés. Écrire un programme où l'utilisateur pense à un nombre entier entre 1 et 100 et l'ordinateur fait des propositions. L'utilisateur répond le caractère M (moins), P (plus) ou T (trouvé), selon que le nombre proposé est respectivement plus grand, plus petit ou égal au nombre inconnu.

### 3.13 Évaluation d'un polynôme de degré $n$

Écrire un programme permettant de saisir la valeur d'une inconnue  $X$  puis les coefficients  $a_0, a_1, \dots, a_{d-1}, a_d$  d'un polynôme de degré  $d$  ( $d$  aura été saisi avant les coefficients), afin d'afficher la valeur en  $X$  du polynôme  $a_0 + a_1X + \dots + a_{d-1}X^{d-1} + a_dX^d$ .

Exemple d'exécution :

```
X = 2
d = 2
a0 = 1
a1 = 2
a2 = 3
Résultat : 17
```

En quoi la saisie de  $X$  avant ou après le polynôme change les possibilités d'écriture de l'algorithme ?

## 4 Tableaux

### 4.1 Permutations de $n$ valeurs

1. Écrire un programme qui effectue une permutation circulaire vers la gauche des éléments d'un tableau (le  $i$ -ème est placé en  $(i - 1)$ -ème place, et le premier élément se retrouve dernier). Les valeurs permutées seront  $n$  nombres entiers préalablement saisis par l'utilisateur.
2. Compléter le programme afin de permuter les valeurs vers la droite, et ainsi faire revenir les valeurs à leur position initiale.

### 4.2 Tableau d'éléments uniques

Écrire un programme permettant de saisir des entiers dans un tableau. Le programme se termine dès qu'une même valeur est saisie deux fois.

---

<sup>2</sup>À propos de la génération de nombres aléatoires :

– `rand()` de la bibliothèque `stdlib` permet de générer un nombre (pseudo-)aléatoire entier, donc `rand()%n` génère un nombre entre 0 et  $n - 1$ .

– Afin d'initialiser la suite aléatoire et ainsi obtenir des exécutions différentes du même programme, il convient d'exécuter au préalable l'instruction `srand(time(NULL))` (une seule fois, par exemple en début de programme). Cela nécessite l'utilisation de la bibliothèque `ctime` en plus de `stdlib`.

### 4.3 Séries miroirs

Écrire un programme qui après avoir effectué la saisie de deux séries de  $n$  nombres entiers, indique si l'une est l'inverse de l'autre. Exemple d'exécution :

```
Longueur des séries de nombres : 5
Première série : 3 8 12 0 7
Seconde série : 7 0 12 8 3
Il s'agit de séries miroirs.
```

### 4.4 Le plus proche [TP]

Écrire un programme qui demande dans un premier temps à l'utilisateur de saisir  $N$  nombres réels ( $N$  étant une constante). L'utilisateur saisira ensuite un nombre de référence, puis le programme affichera quel nombre, parmi les  $N$  saisis préalablement, est le plus proche de la référence.

### 4.5 Histogrammes [TP]

1. Écrire un programme permettant de saisir une suite de notes comprises entre 0 et 20 (la saisie s'arrête pour toute valeur hors des bornes), tout en calculant le nombre d'occurrences de chacune des notes. (On utilisera un tableau `Nb` tel que `Nb[i]` représente le nombre d'occurrences de la note  $i$ ). Afficher ensuite le tableau `Nb` sous forme d'un histogramme horizontal.

Par exemple, 

1	3	0	5	...
0	1	2	3	...

 sera représenté sous la forme

0	:	*
1	:	***
2	:	
3	:	*****

2. ★ Modifier le programme pour afficher un histogramme vertical :

				*
*				*
*				*
*	*	*	*	...
0	1	2	3	...

## 5 Chaines de caractères

### 5.1 Recherche du nombre d'occurrences d'un caractère

Écrire un programme qui détermine le nombre d'occurrences d'un caractère dans une chaîne (le nombre de fois où le caractère apparaît). La chaîne ainsi que le caractère à rechercher seront demandés à l'utilisateur.

### 5.2 Espaces dans les chaînes

Écrire un programme qui demande une chaîne de caractères à l'utilisateur, puis qui crée et affiche trois chaînes dérivées :

1. Aération : on insère un caractère espace après chaque caractère, quel qu'il soit.
2. Compression : on supprime tous les caractères espace.
3. Correction : on supprime tous les caractères espace consécutifs à l'exception du premier. Voici un exemple d'exécution du programme :

```
Entrez votre texte : Sur les bords de l'Isere
S u r   l e s       b o r d s           d e   l ' I s e r e
Surlesbordsdel'Isere
Sur les bords de l'Isere
```

### 5.3 Palindrome [TP]

1. Écrire un programme qui demande à l'utilisateur un mot et qui teste si le mot saisi est un palindrome (mot pouvant être lu dans les deux sens).
2. Étendre le programme afin de tester les palindromes de phrases (phrases pouvant être lues dans les deux sens, en ignorant les espaces).

### 5.4 Jeu du pendu [TP]

Dans ce programme, un premier utilisateur doit préalablement saisir un mot mystère qu'un second utilisateur devra découvrir. Pour cela, ce dernier saisira une première lettre. Chaque occurrence de cette lettre dans le mot mystère sera dévoilée. Le processus est répété jusqu'à ce que le mot soit entièrement découvert. On indique alors le nombre d'essais réalisés.

Exemple de déroulement du jeu (sans effacer la solution) :

```
Entrez votre mot mystère: BONJOUR
Mot mystère: _ _ _ _ _
Entrez une lettre: J
Après 1 essai, voici le mot mystère: _ _ _ J _ _ _
Entrez une lettre: E
Après 2 essais, voici le mot mystère: _ _ _ J _ _ _
Entrez une lettre: O
...
```

→ Pour effacer le mot mystère après l'avoir saisi, il faut utiliser l'instruction `std::system("clear")` de la librairie `cstdlib` qui permet d'effacer l'écran du terminal Linux (ou `std::system("CLS")` pour un terminal Windows).

### 5.5 ★ Anagrammes [TP]

Écrire un programme qui demande de saisir deux chaînes de caractères, puis qui affiche si la deuxième chaîne est une anagramme de la première (i.e. formée des mêmes lettres exactement). Par exemple, CHIEN est une anagramme de NICHE, mais LIVRER n'est pas une anagramme de VRILLE.

## 6 Tableaux à deux dimensions

### 6.1 Matrice carrée

On représente une matrice entière par un tableau à deux dimensions, déclaré comme un tableau de tableau d'entiers.

1. Écrire un programme qui demande la taille d'une matrice carrée à l'utilisateur (nombre identique de lignes et de colonnes), puis numérote en conséquence les cases d'une variable de type matrice. On affichera ensuite les valeurs de la matrice.
2. Modifier la matrice en la remplaçant par sa transposée (lignes et colonnes échangées), sans procéder à une renumérotation.

## 6.2 \* Génération d'une grille de bataille navale [TP] ⑤

L'exercice consiste à générer aléatoirement une grille carrée à deux dimensions, de variables entières, et de taille 10x10. On demande à l'utilisateur le nombre de bateaux, ainsi que la taille de chacun. Les bateaux seront numérotés à partir de 1, et leurs tailles seront mémorisées dans un tableau d'entiers (on n'utilisera pas la première case d'indice 0 du tableau). Ces bateaux seront placés aléatoirement sur la grille, tels qu'ils ne doivent ni sortir de la grille, ni se chevaucher. Une case de la grille valant 0 indique l'absence de bateau ; sinon, le nombre indique le numéro du bateau qui occupe cette case. On affichera ensuite la grille de bataille navale.

Exemple d'exécution :

```
Nombre de bateaux : 5
Longueur du bateau no.1 : 5
Longueur du bateau no.2 : 4
Longueur du bateau no.3 : 3
Longueur du bateau no.4 : 3
Longueur du bateau no.5 : 2
Grille :
0000010000
0000010000
0300010020
0300010020
0300010020
5500000020
0000000000
0000000440
0000000000
0000000000
```