

# Algorithmie en langage C

---

Semestre 3



# Table des matières

<b>1</b>	<b>Paradigmes de programmation</b>	<b>4</b>
1.1	Programmation fonctionnelles . . . . .	4
1.2	Programmation déclarative . . . . .	4
1.3	Programmation Impérative . . . . .	5
<b>2</b>	<b>Programmation impérative en C</b>	<b>6</b>
2.1	Description de l'organisation des données en mémoire . . . . .	6
2.2	Code syntaxe . . . . .	6
2.3	Structure d'une programme en C . . . . .	9
2.4	La compilation . . . . .	9
<b>3</b>	<b>Exercice</b>	<b>11</b>
3.1	Étape 1 : Analyser le problème . . . . .	11
3.2	Étape 2 : Spécifier les sous-problèmes . . . . .	11
3.3	Étape 3 : Le code . . . . .	12
<b>A</b>	<b>Glossaire</b>	<b>13</b>
<b>B</b>	<b>Liste des codes sources</b>	<b>14</b>

# Paradigmes de programmation

Un paradigme est une manière de programmer, il en existe plusieurs :

- La programmation fonctionnelles (cf. 1.1)
- La programmation déclarative (cf. 1.2)
- La programmation impérative (cf. 1.3)

## 1.1 Programmation fonctionnelles

**Type de langage** <sup>1</sup> ou interprétés <sup>2</sup>. Ce paradigme

**Entité de base** Appel de fonction

**Structure de contrôle** Approche récursive.

Elle est utilisée pour des systèmes critiques <sup>3</sup>. Elle à une approche très mathématiques, ce qui permet d'avoir des outils de preuves générique.

Elle possède une abstraction de l'environnement d'exécution, approche détachée de la machine, pas de notion de mémoire.

**Ex** Le Caml est un langage de programmation fonctionnelle

## 1.2 Programmation déclarative

**Type de langage** Interprété

**Entité de base** Règles de déduction logique.

**Structure de contrôle** Possède une abstraction de la machine cible.

**Ex** Le prolog est un langage de programmation déclarative

---

1. Traduction du langage source vers le langage cible(compilation) + une édition de liens, qui est une instanciation sur la machine d'exécution (Recherche d'adresse, mémoire, résolution de fonctions) Elle peut être statique ou dynamique. Ex : C, Adda

2. Le langage source est traduit en langage cible à la volée par un interpréteur. Il est ainsi possible de modifier le programme pendant le fonctionnement du programme.

3. Besoin d'une sureté de fonctionnement

## 1.3 Programmation Impérative

La programmation est directement liée à la machine d'exécution.

**Type de langage** Compilé ou Interprété

**Entité de base** Affectation d'une valeur à une variable, qui est une place en mémoire.

**Structure de contrôle** Séquence, sélection, répétition.

Ex C, Python, Ada ...

# Programmation impérative en C

Énormément de langage sont fondés sur la syntaxe du langage C.

Il a été développé dans les années 1960 par Dennis Ritchie.

On trouvera toujours une partie description de l'organisation des données en mémoire<sup>1</sup>, nous aurons donc une déclaration de variables et un type de données.

```
1 type nomVariable;
```

Listing 2.1 – Syntaxe de déclaration de variable

## 2.1 Description de l'organisation des données en mémoire

Le C possède différents type de données :

**int** Entiers signés

**unsigned int** Entiers non signés

**float** Nombre réel sur 32bits.

**double** Nombre réel sur 64bits.

**char** Entier signé sur 8bits.

**pointeur** type\* ptr ; La case mémoire contient une adresse.

## 2.2 Code syntaxe

### 2.2.1 Blocs

```
1 bloc { // début du bloc  
2 } //fin du bloc
```

Listing 2.2 – Syntaxe d'un bloc

Toute variable est visible dans son bloc de déclaration et ses blocs imbriqués.

Un bloc transforme une séquence en action.

---

1. C'est un grand tableau découpé en cases mémoire.

## 2.2.2 Séquence

```

1 action 1;
2 action 2;
3 action 3;

```

Listing 2.3 – Syntaxe des actions

## 2.2.3 Sélection

```

1 if(condition) {
2     action 1;
3 } else {
4     action 2;
5 }

```

Listing 2.4 – Syntaxe d’une structure de contrôle

Condition est une expression booléenne<sup>2</sup>

## 2.2.4 Répétition

```

1 while(condition) {
2     action;
3 }

```

Listing 2.5 – Syntaxe de répétition

Condition est une expression booléenne, tant que la condition est vrai, les actions se répètent.

## 2.2.5 Affectation

```

1 variable = expression;

```

Listing 2.6 – Syntaxe d’une affectation

## 2.2.6 Opérateurs de base sur les types

= Affectation

+, -, /, \* Opérateurs arithmétiques.

&&, ||, ! Opérateurs logiques

==, !=, <, >, <=, >= Opérateurs booléens

++i, i++, --i, i-- Opérateur unaires d’incrément.

---

2. Expression renvoyant vrai( $\neq 0$ ) ou faux( $= 0$ )

## 2.2.7 Opérateurs d’entrées / sorties

### Ecriture

```
1 printf('format', var1, var2);
```

Listing 2.7 – Syntaxe de l’appel de printf

La chaine format peut contenir une chaine de caractères, avec des caractères spéciaux :


- '%d' Entier sous forme décimale
- '%ox' Entier sous forme hexadécimale
- '%f' Flottant
- '%c' Caractère
- '%s' Chaine de caractères
- '\n' Vide le buffer et fait un retour chariot
- '\t' Tabulation
- '\r' Revient en début de ligne.
- '...' RTFM

Les différents formats doivent être dans l’ordre des variables passés en paramètres.

### Lecture

```
1 scanf('format', &var1); // & représente l'adresse de la variable  
    dans laquelle écrire.
```

Listing 2.8 – Syntaxe de l’appel de scanf


 L’utilisation de cette fonction est risquée. En effet, un utilisateur malveillant peut écrire à des cases mémoires où il n’est pas autorisé.

## 2.2.8 Tableaux

Un tableau est une collection d’éléments de même type.

```
1 // avec tableau le nom de la variable et N la taille du tableau.  
2 type tableau[N], i;  
3 i = tableau[P]; //i recoit la valeur de la case P du tableau
```

Listing 2.9 – Syntaxe de déclaration d’un tableau

 Un tableau commence toujours à 0 et finit à N-1, ainsi, il faut faire très attention au dépassement de la taille d’un tableau.



## 2.2.9 Les sous-programmes

Un sous-programme est un sous-ensemble du programme dans sa hiérarchie fonctionnelle. En C, il correspond toujours à une fonction ou une procédure.

```

1 typeRetour nomFonction (typeArg1 nomArg1, typeArg2 nomArg2) {
2     /*
3         *   code
4     */
5     [return (valeur)];
6 }
```

Listing 2.10 – Syntaxe d'un sous programme

`typeRetour` peut posséder comme valeur les même types qu'une variable, voir 2.1 page 6. Celui-ci peut également être `void`, cela signifie que la fonction ne renvoie rien, c'est donc une procédure.

## 2.3 Structure d'un programme en C

### Interface

- Déclaration des fonctions (prototype)
- Constantes, types
- Comment utiliser le programme  
⇒ Écrire dans un fichier `.h` (header)
- Préprocesseur (`define`, macros, ...)

programme en C ne possède qu'un seul point d'entrée : une instruction est exécutée, c'est la fonction `main`.

### Implantation

- Définitions des fonctions : le code  
⇒ Écrire dans un fichier `.c`

```

1 int main (int argc, char **argv);
2 //le programme renvoie un entier. C'est le profil d'une fonction.
```

Listing 2.11 – Point d'entrée du programme: le main

## 2.4 La compilation

### 2.4.1 Étape 1 : Le préprocesseur

Les préprocesseurs sont les instructions situées en dehors d'un programme, ceux-ci sont préfixés par un dièse (`#`).

**Entrée** fichier `.c`

**Sortie** fichier obtenu une fois les modifications effectués.

```

1 #include // remplace par le contenu du fichier inclus
2 #define Arg1 Arg2 // remplace syntaxique de Arg1 par Arg2
```

Listing 2.12 – Exemple d'instructions pré-processeurs

## 2.4.2 Etape 2 : La compilation

**Entrée** fichier.c, fichier.h

**Sortie** fichier.o

```
1 gcc -c fic1.c fic2.c fic3.c # Créé les fichiers .c
2 gcc *.o nomExe #Créer l'executable.
```

**R** La compilation sera étudiée en détails lors des cours de L3 et M1

---

# Exercice

Écrire un programme qui lit une série de 10 valeurs et affiche la position du minimum et du maximum de la série.

## 3.1 Étape 1 : Analyser le problème

1. Lire les valeurs
2. calculer les min et max
3. afficher le résultat

## 3.2 Étape 2 : Spécifier les sous-problèmes

Identifier les entrée, les sorties et leurs propriétés.

### 3.2.1 LireLesValeurs

**Entrée** Nombre, les valeurs à lire

**Sortie** Tableau contenant les valeurs lues

### 3.2.2 CalculerMinEtMax

**Entrée** Le tableau des valeurs et le nombre de valeur

**Sortie** Position, min et max.

### 3.3 Étape 3 : Le code

```

1  #include <stdlib.h>
2  #define N 100
3
4  void read (int nb, int* tab) ;
5  void calculerMinMax(int nb, int* t, int *pmin, int *vmin, int *pmax
   , int *vmax);
6  void read (int nb, int* tab) ;
7
8  int main (int argc, char** argv) {
9      int pmin, vmin, pmax, vmax;
10     int tab[N];
11     read(10, tab);
12     calculerMinMax(10, tab, &pmin, &vmin, &pmax, &vmax);
13     printf(...);
14 }
15 // un tableau est un pointeur sur le premier élément
16 // peut aussi être écrit int tab[N]
17 void read (int nb, int* tab) {
18     int i;
19     for(i=0; i < nb ; ++i) {
20         scanf('%d', tab+i);
21     }
22 }
23
24 void calculerMinMax(int nb, int *tab, int *pmin, int *vmin, int *
   pmax, int *vmax) {
25     *pmin = 0;
26     *pmax = 0;
27     *vmin = t[pmin];
28     *vmax = t[pmax];
29
30     for(; --nb = b > 0;) {
31         if(tab[nb] < *vmin) {
32             *vmin = tab[nb];
33             *pmin = nb;
34         }
35         if(tab[nb] > *vmax) {
36             *vmax = tab[nb];
37             *pmax = nb;
38         }
39     }
40 }

```

Listing 3.1 – Code du programme

---

# Glossaire

**Compilation** Un compilateur est un programme informatique qui transforme un code source écrit dans un langage de programmation (le langage source) en un autre langage informatique (le langage cible).

**Interprétation** Analyse, traduit et exécute un programme écrit dans un langage informatique. De tels langages sont dits langages interprétés.

L'interpréteur est capable de lire le code source d'un langage sous forme de script, habituellement un fichier texte, et d'en exécuter les instructions après une analyse syntaxique du contenu. Généralement ces langages textuels sont appelés des langages de programmation. Cette interprétation conduit à une exécution d'action ou à un stockage de contenu ordonné par la syntaxe textuelle.

**Édition de liens** Lors d'un développement informatique, l'édition des liens est un processus qui permet de créer des fichiers exécutables ou des bibliothèques dynamiques ou statiques, à partir de fichiers objets.

---

# Liste des codes sources

2.1	Syntaxe de déclaration de variable . . . . .	6
2.2	Syntaxe d'un bloc . . . . .	6
2.3	Syntaxe des actions . . . . .	7
2.4	Syntaxe d'une structure de contrôle . . . . .	7
2.5	Syntaxe de répétition . . . . .	7
2.6	Syntaxe d'une affectation . . . . .	7
2.7	Syntaxe de l'appel de printf . . . . .	8
2.8	Syntaxe de l'appel de scanf . . . . .	8
2.9	Syntaxe de déclaration d'un tableau . . . . .	8
2.10	Syntaxe d'un sous programme . . . . .	9
2.11	Point d'entrée du programme : le main . . . . .	9
2.12	Exemple d'instructions pré-processeurs . . . . .	9
3.1	Code du programme . . . . .	12