

Algorithmie en langage C

Semestre 3

1 Paradigmes de programmation

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

1.1 Programmation fonctionnelles

Type de langage ¹ ou interprétés². Ce paradigme

Entité de base Appel de fonction

Structure de contrôle Approche récursive.

Elle est utilisée pour des systèmes critiques³. 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

1.3 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 ...

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

2 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 mémoire⁴, nous aurons donc une déclaration de variables et un type de données.

```
1 type nomVariable;
```

Listing 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 – Syntaxe d'un bloc

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

Un bloc transforme une séquence en action.

2.2.2 Séquence

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

Listing 3 – Syntaxe des actions

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

2.2.3 Sélection

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

Listing 4 – Syntaxe d’une structure de contrôle

Condition est une expression booléenne⁵

2.2.4 Répétition

```
1 while(condition) {  
2     action;  
3 }
```

Listing 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;  
2 \end{itemize}
```

Listing 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.

2.2.7 Opérateurs d’entrées / sorties

Ecriture

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

Listing 7 – Syntaxe de l’appel de printf

5. Expression renvoyant vrai(! = 0 ou faux(= 0))

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


- '%d' Entier sous forme décimale
- '%ox' Entier sous forme hexadécimale
- '%f' Flottant
- '%c' Caractère
- '%s' Chaîne 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 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 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 10 – Syntaxe d'un sous programme

`typeRetour` peut posséder comme valeur les même types qu'une variable, voir 2.1 page 3. Celui-ci peut également être `void`, cela signifie que la fonction ne renvoie rien, c'est donc une procédure.

2.3 Structure d'une 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'une seul point d'entrée : une instruction est exécuté, 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 11 – Point d'entrée du programme: le main

2.4 La compilation

2.4.1 Étape 1 : Le préprocesseur

Le pré processeur sont les instructions situés en dehors d'un programme, ceux-ci sont préfixé 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 12 – Exemple d'instructions pré-processeurs

2.4.2 Étape 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'exécutable.
```

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

3 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
6     , int *vmax);
7 void read (int nb, int* tab) ;
8
9 int main (int argc, char** argv) {
10     int pmin, vmin, pmax, vmax;
11     int tab[N];
12     read(10, tab);
13     calculerMinMax(10, tab, &pmin, &vmin, &pmax, &vmax);
14     printf(...);
15 }
16 // un tableau est un pointeur sur le premier élément
17 // peut aussi être écrit int tab[N]
18 void read (int nb, int* tab) {
19     int i;
20     for(i=0; i < nb ; ++i) {
21         scanf('%d', tab+i);
22     }
23 }
24
25 void calculerMinMax(int nb, int *tab, int *pmin, int *vmin, int *
26     pmax, int *vmax) {
27     *pmin = 0;
28     *pmax = 0;
29     *vmin = t[pmin];
30     *vmax = t[pmax];
31
32     for(; --nb = b > 0;) {
33         if(tab[nb] < *vmin) {
34             *vmin = tab[nb];
35             *pmin = nb;
36         }
37         if(tab[nb] > *vmax) {
38             *vmax = tab[nb];
39             *pmax = nb;
40         }
41     }
42 }
```

Listing 13 – Code du programme