Langage C – Memo

Chapitre 3 : Les Entrées-Sorties stdio.h

3.1 Qu’est ce qu’une bibliothèque

Une bibliothèque est un ensemble d’’outils ajouté à un langage de programmation. Chaque bibliothèque est dédiée à un thème particulier. Exemple :

- math.h contient des fonctions mathématique de calcul numérique. Elle contient également des constantes telle que M_PI pour la valeur de π
- time.h introduit des fonctions de gestion des durées (addition de dates, gestion du temps d’exécution d’un programme)
- stdio.h contient des fonctions permettant de gérer les entrées-sorties d’un programme. C’est la contraction de Standard Input Output

Comme on peut le constater, les bibliothèques sont des fichiers .h. Cela signifie que ce sont des fichiers d’en tête (header). Ils ne contiennent que les définitions des fonctions C décrites dans un fichier .c

Pour utiliser une libraire, une utilise la directive

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

3.2 Affichage de données

3.2.1 Affichage de texte

Il existe 2 fonctions pour afficher du texte :

- putchar, qui prend en paramètre un caractère unique à afficher.
- puts, qui affiche une chaîne de caractères. Une chaîne est déclarée à l’aide de double cotes, pour un caractère, ce sont les simples cotes que l’on utilise.

3.2.2 Affichage d’autres données

Il existe une fonction plus générique pour l’affichage en console. Par défaut, celle-ci affiche une chaîne de caractères, mais en utilisant des caractères spéciaux, on peut afficher d’autres éléments. Exemple :

```
printf("Ceci est un entier: %d", 51);
```

La ligne ci dessus affichera en console :

```
Ceci est un entier: 51
```

Le %d a été remplacé par le nombre 51, qui est de type int.

Pour chacun des types, il existe un caractère spéciale pour l'affichage dans un printf.

Caractère dans la chaîne	Type affiché
%c	Type char
%d	Type int
%f	Type float
%lf	Type double
%.?f	Type float où le ? indique le nombre de chiffre derrière la virgule (ex. %.5f → 5 chiffres après la virgule)
%e	Type float sous forme de puissance de 10
%g	Type float, avec un format « passe partout »
%s	Type String

3.3 Lecture au clavier

Pour lire une valeur saisie au clavier, il faut utiliser la fonction scanf qui prend en argument un type de donnée (comme pour le printf), et une adresses de variable (nous verrons cela dans un prochain chapitre).

```
Int maVariable;

scanf("%d", &maVariable);
```

Dans le code précédent, on a déclaré une variable de type int. A l'aide du scanf, on va affecter une valeur saisie par l'utilisateur. Comme le scanf utilise comme deuxième argument une valeur et non directement une variable, on utilise le & en début du nom de la variable. Pour le premier argument, les formats autorisés sont les suivants :

Caractère dans la chaîne	Type affiché
%d	Type int
%u	Type unsigned int
%f	Type float
%lf	Type double
%c	Type char

Chapitre 4: Les structures de contrôle

4.1 Différence entre instruction et directive

En C une instruction est terminée par un ; et indique une nouvelle étape de l'algorithme. Pour des raison de lisibilité, on évite de mettre plusieurs instructions par ligne et l'on ne scinde une instruction sur plusieurs ligne que si celle-ci est trop longue (+ de 80 caractère)

Une directive est généralement placée en début de programme, sert principalement à déclarer des élément qui vont être utile à l'algorithme, et s'écrit avec un # en début de ligne et ne possède pas de ; en fin de ligne.

4.2 Expressions et opérateurs

4.2.1 Expression

Une expression est tout ce qui représente une valeur numérique. Une expression simple est une affectation d'une valeur à une variable ou une constante, et expression complexe est une affectation d'une valeur à une variable par le biais d'un calcul mathématique.

4.2.2 Opérateurs

Un opérateur décrit une action effectuée sur plusieurs opérandes. Toutes lignes utilisant un opérateur est une expression

L'opérateur affectation

Permet d'affecter une valeur à une variable. C'est le symbole =

Les opérateurs mathématiques

Les opérateurs mathématiques sont les 5 opérations mathématique de base : + (addition), - (soustraction), * (multiplication), / (division) et % (modulo), mais également les opérateurs unaires d'incrémentation : ++ ; et de décrémentation : – ;

```
int a = 0;

a++; //Après cette instruction, a vaudra 1
```

Les règles de priorités pour ces opérateurs sont les suivantes :

- Priorité 1 : les opérateurs unaire
- Priorité 2 : la division, la multiplication et le modulo
- Priorité 3 : les additions et soustractions

Les opérateurs de comparaison

Ils servent à comparer des valeurs entre elles et renvois une valeur booléenne True ou False.

- > → Supérieur
- < → Inférieur
- >= → Supérieur ou égale
- <= → Inférieur ou égale
- != → Différent

- == → égale (à ne pas confondre avec le = de l'affectation!)

Les opérateurs de logique

Ils permettent de réaliser des opérations sur des valeurs booléennes

- && → conjonction ET
- || → conjonction OU
- ! → négation

```
(5==5) && (6!=2) // Cette expression vaut True
```

4.3 Les structures conditionnelles

4.3.1 L'alternative composé

Il s'agit d'une simple Si Alors Sinon. En C, il se rédige comme ceci:

```
if (expression booléenne)
   instruction1;
else
   instruction2;
```

Si l'expression booléenne renvoi True, alors on exécute l'instruction 1, sinon, on exécute l'instruction 2.

Si l'on souhaite exécuter plusieurs instructions dans une structure conditionnelle, on doit encadrer ces instructions par des {}

4.3.2 L'opérateur ternaire

En C, on peut réaliser une affectation combiner à une structure alternative en une seule instruction à l'aide de l'opérateur ternaire. La syntaxe est la suivante :

```
a = b ? 5 : 10;
```

Cette ligne signifie que si a est égale à b, alors a prendra pour valeur 5, sinon, a prendra pour valeur 10.

4.3.3 L'alternative SWITCH

Cela permet de tester plusieurs valeurs pour une expression et évite les structures composées imbriquées les unes dans les autres.

La syntaxe est la suivante :

```
int choix;
… // la variable choix va changer dans cette instruction
```

```
switch(choix) {
   case 1: puts("choix vaut 1");
      break;
   case 2: puts("choix vaut 2");
      break;
   default: puts("choix vaut autre chose");
}
```

La valeur de la variable choix va être comparée successivement à la valeur indiquée après le mot clef case. Si elle ne correspond à aucune valeur, le traitement effectué sera celui décrit après le mot clef default.

4.4 Les structures itératives

4.4.1 L'instruction for

La boucle for permet de répéter un nombre de fois fixée, une suite d'instructions. Sa syntaxe est la suivante :

```
for (int i = 0; i < 10; i++) {
   instruction1;
   …
   instructionN;
}
```

Dans l'instruction for, on déclare un itérateur (une variable qui va évoluer pendant la boucle), puis on déclare une condition de sortie sur l'itérateur (si celui-ci atteint une valeur, on arrête la boucle for), et enfin, on indique comment la valeur de l'itérateur change à chaque itération.

Les instructions qui sont répétés durant les boucles sont encadrées par les {}

4.4.2 L'instruction while

La boucle while est une bloque d'instruction qui est exécutée tant que la condition spécifiée est vrai.

Sa syntaxe est la suivante:

```
while (condition) {
   instruction;
}
```

La condition est similaire à une condition trouvable dans une alternative composée. Attention, il faut que les paramètre de la condition change dans la boucle while si l'on ne veut pas entrer dans une boucle infinie !

De plus, si la condition est d'entrée fausse, aucune instruction de la boucle ne sera executée.

4.4.3 L'instruction do-while

Cette instruction d'itération est similaire au while vu précédemment, à la nuance que le code contenu dans la boucle est exécuté au moins une fois, qu'importe que la condition soit vrai ou fausse. En effet, le test est effectué après les instructions du bloc itéré. La syntaxe est la suivante :

```
do {
   instruction;
} while (condition);
```