

Cours de C : Partie 1

Introduction

Ce cours n'a pas la vocation d'apprendre des notions poussées dans le langage C mais il peut plus servir de mémento pour les débutants. Il y a l'essentiel des bases pour l'instant.

Sommaire

- I - Variables
- II - Opérateurs et structures de contrôles
- III - Fonctions
- IV - Les structures
- V - Tableaux

I - Variables

Définition : Une variable permet de stocker des informations (des données) comme des nombres, des caractères, et des chaînes de caractères).

Pour utiliser une variable il faut la déclarer :

Syntaxe : `<type> maVariable;`

Exemple : `int maVariable;`

Pour la remplir (après la déclaration) :

`maVariable = 10;`

Les types

Type	Définition
int	Entier
char	Caractère
float	Nombre à virgule flottante (Simple précision)
double	Nombre à virgule flottante (Double précision)
short	Entier court
long	Entier long

Tout ces types peuvent contenir des valeurs positives comme négatives.

Pour avoir des nombres uniquement positifs, il faut utiliser le mot-clé unsigned

Syntaxe : `unsigned <type> maVariable;`

II - Opérateurs et structures de contrôles

1. Opérateurs

Opérateur d'affectation : =

Syntaxe : maVariable = uneValeur;

Opérateurs mathématiques :

Opérateur	Symbole utilisé	Exemple
Addition	+	i + j
Soustraction	-	i - j
Multiplication	*	i * j
Division	/	i / j
Modulo	%	i % j
Incréméntation	++	i++ (ajoute 1 à la valeur de i)
Décrémentation	--	i-- (enlève 1 à la valeur de i)

Opérateurs de comparaison :

Opérateur	Symbole utilisé	Exemple
Egalité	==	i == j
Supérieur	>	i > j
Inférieur	<	i < j
Supérieur ou égal	>=	i >= j
Inférieur ou égal	<=	i <= j
Différent de	!=	i != j

2. Structures de contrôles

Structures de contrôle (si)

Syntaxe :

```
if (expression)
    instruction;
```

Si vous avez plusieurs instructions, vous devez créer un bloc avec les accolades :

```
if (expression)
{
    instruction 1;
    instruction 2;
}
```

Exemple d'utilisation :

```
if (x >= y)
{
    x++; /* On incrémente de 1 la variable x */
}
```

Par convention, le contenu d'une instruction (ici le if) sera décalé d'un caractère de tabulation, ce n'est que pour une meilleure lisibilité, on appelle cela l'indentation.

Structures de contrôle else (sinon)

Syntaxe :

```
else
    instruction;
```

Si vous avez plusieurs instructions, il faut les délimiter par les accolades pour signifier un bloc.

Exemple d'utilisation :

```
if (x > y)
    x++;
else
    y++;
```

Le Sinon-si

Dans certains langages il existe la structure de contrôle sinon-si (elseif), en langage C ce n'est pas explicite, la manière de faire est la suivante :

Exemple d'utilisation :

```
if (x > y)
    x++;
/* Sinon */
else
    /* Si */
    if (x == y)
        x=0;
    else
        y++;
```

Structures de contrôle switch

Syntaxe :

```
switch (variable)
{
    case 'valeur1' : instruction;
    case 'valeur2' : instruction2;
    default : instruction3 ;
}
```

Ici si la variable correspond à une des valeurs données, l'instruction correspondante sera exécutée , sinon il exécutera l'instruction définit dans la valeur par défaut.

Ici dans la syntaxe, lorsqu'une instruction est exécutée, on souhaiteras sortir de la boucle switch, dans ce cas là il faut utiliser l'instruction *break*.

Il existe aussi l'instruction *continue*.

Structures de contrôle while (tant que)

But : Répéter des instructions tant qu'une condition n'est pas remplie.

Syntaxe :

```
while (expression)
instruction;
```

Si vous avez plusieurs instructions, il faut les délimiter par les accolades pour signifier un

bloc.

Exemple d'utilisation :

(Supposons que i est initialisé à 0)

```
while (i < 10)
{
    i++; /* à incrémenter absolument, sinon c'est la boucle infinie ! */
}
```

Structures de contrôle for (pour)

Syntaxe :

```
for ( initialisation ; condition ; incrémentation/décrémentation )
instruction;
```

Si vous avez plusieurs instructions, il faut les délimiter par les accolades pour signifier un bloc.

Exemple d'utilisation :

(Supposons qu'on a déclaré i auparavant)

```
for (i=10 ; i > 0 ; i--)
{
    printf("Bonjour \n");
}
```

On peut dire que while et for sont des boucles, mais quand privilégier l'une ou l'autre.

La structure de contrôle for sert quand on est certains des conditions de départ et de fin de la boucle.

On utilise la structure de contrôle while lorsque l'on ne connaît pas vraiment la fin de celle-ci.

III - Fonctions

En C, on a un programme principal appelé main, sa présence sera vérifié à chaque fois lors de sa compilation.

Lorsque l'on utilise des instructions qui se répète dans le programme, on peut créer une fonction, qui effectuera les instructions souhaités.

Syntaxe :

```
<type_retourne> NomFonction (<type1> argument1, [ <type2> argument2 ] )
{
    instruction;
}
```

Une fonction peut exécuter un ensemble d'instruction et on peut avoir besoin de retourner une valeur, c'est pourquoi on définit le type retourné.

Lorsqu'une fonction ne retourne rien, on met comme type retournée : void.

Dans la fonction on peut avoir besoin de données extérieur à celle-ci, c'est pourquoi on a des

arguments après le nom de la fonction, également appelé paramètres.

Exemple d'une fonction :

```
int MaFonction (char param1, int param2)
{
    int maVariableLocale;
    maVariableLocale = param2;
    return maVariableLocale;
}
```

Ici la fonction retourne un entier (int), on a déclaré dans cette fonction une variable dites locale, car elle n'existera que dans cette fonction.

On utilise également un des paramètres donnés (param2) que l'on pourra utiliser partout dans la fonction.

En dernier nous retournons la valeur de maVariableLocale.

Utilisation de la fonction dans un programme :

Syntaxe :

NomFonction (argument1, argument2);

Ici dans notre exemple ça donnerai :

```
MaFonction ('a', 21);
```

Comme la fonction retourne une valeur, cette valeur soit on veut la récupérer et la placer dans une variable soit l'afficher.

Pour la récupérer dans une variable :

```
maVariable = MaFonction('a', 21);
```

Un compilateur C lis le code linéairement, c'est-à-dire, de la première ligne à la dernière ligne. Lorsqu'il arrivera au programme principal et si les fonctions sont placées après celui-ci, le compilateur ne saura pas ce que sont ces programmes.

C'est pourquoi on utilise des prototypes.

La syntaxe d'un prototype est simplement la première ligne d'une fonction fermé par un point virgule.

Syntaxe :

<type_retourne> NomFonction (<type> argument1, [<type> argument2]);

IV - Les structures

Une structure contient une ou plusieurs variables, que l'on appelle membre.

Syntaxe d'une structure :

```
struct NomStructure
{
    <type> Membre;
};
```

Ici on a défini une structure que l'on peut utiliser pour déclarer une variable, il existe 2 méthodes :

- 1ère méthode :

```
struct NomStructure
{
    <type> Membre;
}maVariable1;
```

- 2ème méthode :

Une fois qu'on a défini notre structure comme dans la syntaxe, on peut déclarer cette variable structure :

```
struct NomStructure maVariable1;
```

Initialisation de cette variable structure :

```
struct NomStructure maVariable1 = { valeurMembre, etc... }
```

Une fois que l'on a défini, déclaré et initialisé notre variable structure, on peut déclarer l'utiliser comme une variable simple mais en précisant auparavant le nom de la variable structure :

```
maVariable1.Membre;
```

Mot clé Typedef

On utilise le mot clé typedef pour créer un synonyme de la structure.

Syntaxe :

```
typedef struct
{
    <type> NomMembre1;
} NomStructure;
```

Ensuite pour déclarer votre structure :

```
NomStructure maVariable1;
```

Les deux notations (avec ou sans typedef) s'utilisent indifféremment.

Quand utiliser une structure :

- on peut l'utiliser pour " structurer " un ensemble de variable, ce qui permet d'avoir un code plus propre;
- on peut également l'utiliser lorsque l'on veut retourner plusieurs variables d'une fonction, dans ce cas là on retourne la variable structure, ce qui ne pose pas de problèmes.

V - Tableaux

C'est une variable qui regroupe plusieurs éléments du même type.

Syntaxe de la déclaration :

```
<type> nomTableau[];
```

Il faut donner une taille à ce tableau, pour l'instant notre tableau sera dit statique.

On attribue cette valeur lors de la déclaration :

```
<type> NomTableau[taille];
```

Par exemple :

```
int monTableau[5];
```

On pourra donc rentrer 5 valeurs dans notre tableau.

Pour parcourir ce tableau et identifier un élément par rapport à un autre, on donne un indice à une valeur du tableau.

Cet indice est numérique.

Par exemple, on veut connaître la 5ème valeur du tableau :

On utilisera cette notation :

```
NomTableau[4];
```

On donne l'indice 4 car on a 5 valeurs mais la notation commence à partir de 0 et non de 1, si on a 5 valeurs et que l'on démarre de 0, notre 5ème valeur à l'indice 4.

Caractères et tableaux

Lorsque l'on déclare une variable de type char, on la déclare pour un caractère, les tableaux servent à créer des chaînes de caractères.

Par exemple, si on veut rentrer le mot 'Bonjour', il faudra créer un tableau de char de 7 cases, mais en langage C, il y a un caractère d'échappement \0 qui s'ajoute en fin du mot, il nous faudra donc créer un tableau de 8 valeurs.

Affectation des valeurs au tableau :

Soit à l'initialisation du tableau :

Par exemple :

```
int monTableau[3] = { 8, 7, 1 };
```

Soit par affectation directement à l'indice :

Par exemple :

```
int monTableau[2];  
monTableau[1] = 10;
```

Affectation d'une chaîne de caractères :

Deux méthodes à l'initialisation :

```
char maChaine[6] = "Salut";  
ou  
char maChaine[6] = {'S', 'a', 'l', 'u', 't'};
```

On peut remarquer que l'on ne mets pas le caractère d'échappement.

On peut également affecter chaque caractère comme dans l'exemple de l'affectation

directement à l'indice.

Tableaux à plusieurs dimensions :

On peut créer des \"sous-tableaux\" dans des tableaux.

Initialisation :

```
int monTableau[taille1][taille2];
```

La première taille correspond à la taille de chaque emplacement 'principal', la seconde taille correspond au nombre de 'case' allouée à chaque emplacement principal.