Introduction

Le langage C peut être considéré, pour certaines limites, comme un sous-ensemble du langage de programmation C++ - généralement un programme écrit en C est valide en C++, mais pas toujours, car C++ pose dans certains cas plus de restrictions que C (par exemple le contrôle de type des paramètres d'une fonction).

Un exemple d'un programme qui fonctionne dans les deux langages est représenté ci-dessous :

```
#include <stdio.h>

int main() {
        printf("Hello, World!\n");
        return(0);
}
```

1. #include <stdio.h> - donne l'instruction au compilateur pour inclure la déclaration des fonctions de la librairie 'standard input/output' (qui déclare plusieurs fonctions parmi lesquelles la fonction "printf").

2. Le programme ci-dessus défini une fonction appelée main. Chaque programme C ou C++ doit avoir une fonction appelée main, et ce programme commence par exécuter cette fonction.

3. Le corps de la fonction main contient un appel a la fonction printf qui écrit "Hello, World!\n" sur la sortie standard.

4. Un backslash ("\") suivi par un autre caractère désigne un caractère spécial; dans le cas ci-dessus \n désigne le caractère nouveau ligne.

5. La fonction main est du type int et elle retourne la valeur 0 au système d'exploitation.

Le programme suivant (valide seulement pour C++) produit le même résultat que le précédent.

```
#include <iostream.h>

int main() {
        cout << "Hello, World!\n";
        return(0);
}
```

1. Ce programme utilise "streams"; des classes spéciales utilisées pour les entrées et les sorties.

2. **cout** est le stream de la sortie standard.

3. L'opérateur << ("mettre dans") écrit son second argument dans le premier. Dans le cas ci-dessus, la chaîne de caractères "Hello, World! \n" est écrite au stream de la sortie standard **cout**.

Types Fondamentaux (les mêmes pour C et C++)

Mots Clés	**Description**
Char	Caractère (occupe 1 octet)
Int	Entier (occupe 1 mot, 16-bit ou 32-bit dépendant du système d'exploitation)
long int	Entier (occupe 32 bits)
Float	Nombres décimaux – simple précision (6 digits)
Double	Nombres décimaux – double précision (15 digits)

Exemple d'usage des types de base, l'instruction **if** et la saisie avec l'entrée standard (cin) :

```
// Programme qui fait la conversion pouce-au-centimètre
// et centimètre-au-pouce

#include <iostream.h>

int main() {
        const float fac = 2.54;
        float x,in,cm;
        char ch = 'y';

        cout << " entrer la longueur : ";

        cin >> x;   // saisie d'un nombre décimal
        cin >> ch; // saisie d'un suffixe (un caractère)

        if  (ch == 'i') { // pouce
                in   = x;
                cm = x*fac;
        }
        else if (ch == 'c') { // cm
                in = x/fac;
                cm = x;
        }
        else in=cm=0;

        cout << in << "in=" << cm << "cm\n";
        return 0;
}
```

1. fac et ch sont initialisées au temps de déclaration.

2. fac est déclaré comme une constante : tout essai ultérieur à changer sa valeur est erroné.

3. x reçoit un nombre décimal de l'entrée standard **cin**. ch reçoit un caractère de **cin**.

Des intructions équivalentes utilisant la librairie du langage **C** 'stdio.h' sont :

```
scanf("%f ",&f);
scanf("%c",&ch);
```

1. %c – format special indiquant le type de la variable ('char' dans ce cas).
2. &ch - "addresse de" la variable ch.

L'instruction 'if' peut être remplacée par l'instruction 'switch' :

```
Switch(ch) {
        case 'i' :  in  = x;
                    cm  = x*fac;
                    break;
        case 'c' : in   = x/fac;
                    cm  = x;
                    break;
        default  : in=cm=0;
}
```

Types Dérivées (Tous existent en C sauf la référence)

1. * - Pointeur
2. & - Référence
3. ( ) - Fonction
4. [ ] - Tableau

Les Variables Pointeurs *

1. Chaque octet a une adresse unique en mémoire.
2. Une machine avec 16 Megaoctets de mémoire principale possède 16,777,216 octets.
3. Les variables occupent un ou plusieurs octets de la mémoire.

4. Dans l'exemple suivant, la variable i occupe les octets 2000 et 2001, donc l'addresse de i est 2000.

Un pointeur est une variable qui contient l'adresse d'une autre variable.

Exemple :

```
  …
int i = 0;          // supposons i occupe les positions 2000 et 2001
int *p;             // declare un 'pointeur à ' un entier

p = &i;             // p reçoit l'adresse de i (2000 dans notre exemple)
*p = 1;             // l'entier qui commence à la mémoire pointée par p reçoit la valeur 1

cout << " l'adresse est " << p << "\n";
cout << " la valeur de i est " << i << "\n";
  …
```

La sortie sera :

L'adresse est 2000;
La valeur de i est 1;

N.B. Le symbole '&' vu jusque là est l'opérateur "adresse de" ; il ne faut pas le confondre avec '&' la variable de référence.

Variable de Référence '&'

Les références sont déclarés comme des synonymes aux variables

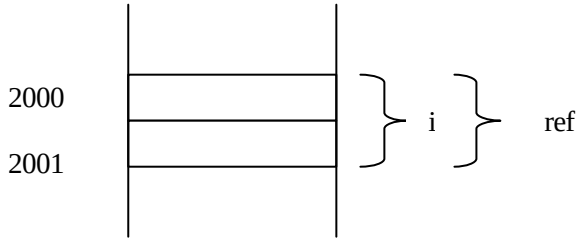
.
.
.

```
int i;
int &ref = i;
```

.
.
.

Les instructions suivantes sont équivalents. Tous les deux incrémentent i :
i++; ou ref++;

i et ref, tous les deux, font référence a la même adresse de mémoire, par exemple :



Fonctions( )

La forme générale d'une fonction :

```
valeur-à-retournée NomFonction (arg1, arg2) {
        <corps de la fonction>
}
```

Exemple 1 : Fonction qui calcule le carré d'un nombre :

```
float CARRÉ (float x) {
        return x*x;
}
```

Exemple 2 :

```
#include <iostream.h>
void FausseIncrementation(int i) {
        i++;
}
void Incrementation(int &i) {
        i++;
}

int main( ) {
        int x=0;
        int y=0;

        FausseIncrementation(x);
        Incrementation(y);

        cout << "x = " << x << " y = " << y << "\n";
        return 0;
}
```

Ce programme va générer la sortie suivante :

x = 0 y = 1

N.B. quand la valeur-à-retournée est 'void' la fonction ne retourne aucune valeur.

1. Dans la fonction FausseIncrementation l'argument est passé par valeur : la valeur de 'x' est copiée dans la variable locale 'i', 'i' est incrémenté mais 'x' reste le même.

2. Dans la fonction Incrementation l'argument est passé par référence : la référence à 'y' est passé à la référence de la variable locale 'i', quand 'i' est incrementé 'y' va être incrementé aussi. 'i' et 'y' vont être des synonymes.

Tableau [ ]

- C'est une collection de n objets de même type, indexés de 0 à n-1.

Exemple :

.
.
.

```
int a[10];
int i;

for ( i=0; i<10; i++ ) {
        a[i] = i + 3;
}

// afficher le tableau dans l'ordre inverse
for ( i=9; i>=0; i--) {
        cout << " Pos " << i << " contient " << a[i] << "\n";
}
```

.
.
.

Result :

	3	4	5	6	7	8	9	10	11	12
a =	0	1	2	3	4	5	6	7	8	9

Sortie

```
Pos 9 contient 12
Pos 8 contient 11
```

.
.
.

```
Pos 0 contient 3
```

Autres exemples de déclaration de tableaux :

```
float v[3];   // tableau à 3 nombres décimaux
              // v[0], v[1], v[2]

int a[2][5]; // deux tableaux à 5 entiers
```

Char* vpc[32]; // tableau à 32 pointeurs à char

Exemples d'initialisation de tableau :

int a[] = {3,4,5,6,7,8,9,10,11,12};

On n'a pas besoin de spécifier la taille, le compilateur assigne 10 positions automatiquemt.

char voyelles[] = "aeiou"; // 5 positions

Noter que le suivant est invalide :

char voyelles[5];
voyelles = "aeiou"; // erreur

Car l'affectation n'est pas définie pour le type de tableau.