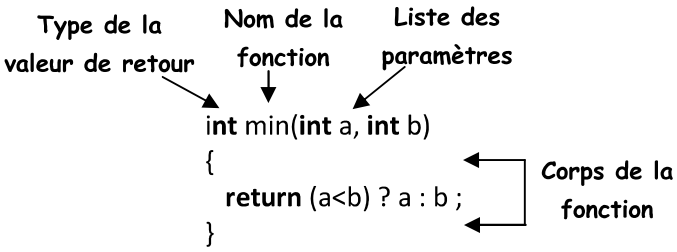
Chapitre III : Fonctions en C++

III.1 Création et utilisation d'une fonction

Une fonction est une opération définie par le programmeur caractérisée par :

- Son nom,
- le type de valeur qu'elle renvoie,
- les paramètres qu'elle reçoit pour faire son travail,
- l'instruction-bloc qui effectue le travail (corps de la fonction).

Exemple:



III. 1.1 Déclaration d'une fonction

Toute fonction doit être déclarée avant d'être appelée pour la première fois. La définition d'une fonction peut faire office de déclaration.

Il peut se trouver des situations où une fonction doit être appelée dans une autre fonction définie avant elle. Comme cette fonction n'est pas définie au moment de l'appel, elle doit être déclarée.

Le rôle des déclarations est donc de signaler l'existence des fonctions aux compilateurs afin de les utiliser, tout en reportant leur définition de ces fonctions plus loin ou dans un autre fichier.

Syntaxe :

```
type nomDeLafonction (paramètres) ;
```

Exemple [4b]***:***

```
double Moyenne(double x, double y);  // Fonction avec paramètres et type retour
char LireCaractere();  //  Fonction avec type de retour et sans paramètres
void AfficherValeurs(int nombre, double valeur); //Fonction avec paramètres et sans type de retour
```

III.1.2 Définition d'une fonction

Toute fonction doit être définie avant d'être utilisée.

Syntaxe:

```
type nomDeLafonction (paramètres)
{
  Instructions ;
}
```

Il est possible de créer *plusieurs fonctions ayant le même nom.* Il faut alors que la liste des arguments des deux fonctions soit différente. C'est ce qu'on appelle la **surcharge** d'une fonction.

Exemple:

```
#include <iostream>
using namespace std;

double moyenne(double a, double b);  // déclaration de la fonction somme :

int main()
{   double x, y, resultat;
    cout << "Tapez la valeur de x : "; cin >> x;
    cout << "Tapez la valeur de y : "; cin >> y;
    resultat = moyenne(x, y);  //appel de notre fonction somme
    cout <<" Moyenne entre "<<  x <<" et "  << y << " est : " << resultat << endl;
 return 0;
}
// définition de la fonction moyenne :
 double moyenne(double a, double b)
{ double moy;
   moy = (a + b)/2;
  return moy;
}
```

- Dans ce programme, on a créé une fonction nommée moyenne qui reçoit deux nombres réels a et b en paramètre et qui, une fois qu'elle a terminé, renvoie un autre nombre moy réel qui représente la moyenne de a et b. l'appel de cette fonction se fait au niveau du programme principale (main).

- Dans cet exemple, le prototype est nécessaire, car la fonction est définie après la fonction main() qui l'utilise. Si le prototype est omis, le compilateur signale une erreur.

- Le prototype peut être encore écrit de la manière suivante:

  double moyenne(**double** , **double**)

Affichage de l'exécution:

```
Tapez la valeur de x : 12.5
Tapez la valeur de y : 5.78
 Moyenne entre 12.5 et 5.78 est : 9.14

Process returned 0 (0x0)   execution time : 34.594 s
Press any key to continue.
```

III.1.3 Fonctions sans retour

C++ permet de créer des fonctions qui ne renvoient aucun résultat. Mais, quand on la déclare, il faut quand même indiquer un type. On utilise le type **void**. Cela veut tout dire : il n'y a vraiment rien qui soit renvoyé par la fonction.

Exemple:

```
#include <iostream>
using namespace std;

void presenterPgm () ;

int main ()
{
  presenterPgm();
  return 0;
}

void presenterPgm ()
{
  cout << "ce pgm …";
}
```

- Une fonction produit toujours au maximum un résultat, c'est-à-dire qu'Il n'est pas possible de renvoyer plus qu'une seule valeur.

III.2 Surcharge des fonctions [4b]

En C++, Plusieurs fonctions peuvent **porter le même nom** si leurs **signatures diffèrent**. La **signature** d'une fonction correspond aux **caractéristiques de ses paramètres**

- leur nombre
- le type respectif de chacun d'eux

Le compilateur choisira la fonction à utiliser selon les paramètres effectifs par rapport aux paramètres formels des fonctions candidates.

Exemple:

```
#include <iostream>
using namespace std;

// définition de la fonction somme qui calcul la somme de deux réels
double somme(double a, double b)
{  double r;
   r = a + b;
  return r;
}
// définition de la fonction somme qui calcul la somme de deux entiers
double somme(int a, int b)
{  double r;
   r = a + b;
   return r;
}
// définition de la fonction somme qui calcul la somme de trois réels
double somme(double a, double b, double c)
{  double r;
   r = a + b + c;
   return r;
}
```

```
int main()
{  double x, y,z, resultat;
   cout << "Tapez la valeur de x : "; cin >> x;
   cout << "Tapez la valeur de y : "; cin >> y;
   cout << "Tapez la valeur de z : "; cin >> z;

//appel de notre fonction somme (double,double)
  resultat = somme(x, y);
  cout <<  x << " + " << y << " = " << resultat << endl;

//appel de notre fonction somme (int,int)
  resultat = somme(static_cast<int>(x), static_cast<int> ( y));
  cout <<   static_cast<int>(x) << " + " << static_cast<int>( y) << " = " << resultat << endl;

//appel de notre fonction somme (double, double, double)
  resultat = somme(x, y,z);
 cout<<  x << " + " << y << " + " << z<< " = " << resultat << endl;

//appel de notre fonction somme (double, double, double)
  resultat = somme(static_cast<int>(x), static_cast<int>( y), static_cast<int>(z));
 cout<< static_cast<int> ( x) << " + " << static_cast<int>(y) << " + " << static_cast<int>( z)<< " = " <<
resultat << endl;

 return 0;
}
```

Affichage de l'exécution:

```
Tapez la valeur de x : 5.6
Tapez la valeur de y : 7.5
Tapez la valeur de z : 2.3
5.6 + 7.5 = 13.1
5 + 7 = 12
5.6 + 7.5 + 2.3 = 15.4
5 + 7 + 2 = 14
```

III.3 Arguments par défaut

On peut, lors de la déclaration d'une fonction, donner des valeurs par défaut à certains paramètres des fonctions. Ainsi, lorsqu'on appelle une fonction, on ne sera pas obligé d'indiquer à chaque fois tous les paramètres! [4b]

Exemple :

```
#include <iostream>
 using namespace std;

 void afficheLigne(const char c, const int n=5)
{  for(int i(0) ; i<n ;  ++i)
    cout<<c ;
 }

int main()
{  afficheLigne('+') ;
   cout<<endl ;
   afficheLigne('*',8) ;
 return 0 ;
}
```

Affichage de l'exécution:

```
+++++
********
Process returned 0 (0x0)   execution time : 3.354 s
Press any key to continue.
```

✍ Il y a quelques règles que vous devez retenir pour les valeurs par défaut:

- Seul le prototype doit contenir les valeurs par défaut.
- Les valeurs par défaut doivent se trouver à la fin de la liste des paramètres.
- Vous pouvez rendre tous les paramètres de votre fonction facultatifs.

III.4 Passage des paramètres par valeur et par variable

Il y a deux méthodes pour passer des variables en paramètres dans une fonction: le **passage** par **valeur** et le **passage** par **variable**. Ces méthodes sont décrites ci-dessous.

III.4.1 Passage par valeur

La valeur de l'expression passée en paramètre est copiée dans une variable locale. C'est cette variable qui est utilisée pour faire les calculs dans la fonction appelée.

Pour mieux comprendre, Prenons le programme suivant qui ajoute 2 à l'argument fourni en paramètre.

Exemple :

```
#include <iostream>
 using namespace std;

void sp (int );

int main()
{int n = 3;
cout << "n=" << n << endl;
sp (n);
cout << "n=" << n;
return 0 ;
}
void sp (int  nbre)
{   cout << "----------"<<endl ;
   cout << "nbre=" << nbre << endl;
    nbre = nbre + 2;
   cout << "nbre=" << nbre;
   cout << "----------"<<endl ;
}
```

Affichage de l'exécution:

```
n=3
----------
nbre=3
nbre=5
----------
n=3

Process returned 0 (0x0)   execution time : 1.782 s
Press any key to continue.
```

III.4.2 Passage par variable

Consiste à passer l'adresse d'une variable en paramètre. Toute modification du paramètre dans la fonction affecte directement la variable passée en argument correspondant, puisque la fonction accède à l'emplacement mémoire de son argument. Il existe 2 possibilités pour transmettre des **paramètres par variables** :

- Les **références**
- Les **pointeurs**

III.4.2.1 Passage par références

Consiste à passer une référence de la variable en argument. Ainsi, aucune variable temporaire ne sera créée par la fonction et toutes les opérations (de la fonction) seront effectuées directement sur la variable. Le plus simple est d'utiliser le mécanisme de référence « **&** ».

Exemple :

```
#include <iostream>
 using namespace std;

void sp (int  &);  // ajout de la référence au niveau du prototype

int main()
{int n = 3;
cout << "n=" << n << endl;
sp (n); //appel de la fonction
cout << "n=" << n<<endl ;;
return 0 ;
}
void sp (int &  nbre)       //Ajout de la référence dans la fonction
{   cout << "----------"<<endl ;
    cout << "nbre=" << nbre << endl;
     nbre = nbre + 2;
    cout << "nbre=" << nbre<<endl ;
    cout << "----------"<<endl ;
}
```

Affichage de l'exécution:

```
n=3
----------
nbre=3
nbre=5
----------
n=5

Process returned 0 (0x0)   execution time : 1.674 s
Press any key to continue.
```

III.4.2.2 Passage par adresses (pointeurs)

Consiste à passer l'adresse d'une variable en paramètre. Toute modification du paramètre dans la fonction affecte directement la variable passée en argument correspondant, puisque la fonction accède à l'emplacement mémoire de son argument. Le pointeur indique au compilateur que ce n'est **pas la valeur** qui est transmise, **mais une adresse** (un pointeur).

Exemple :

```
#include <iostream>
 using namespace std;

void sp (int  *);  // ajout de l'étoile au niveau du prototype

int main()
{int n = 3;
cout << "n=" << n << endl;
sp (&n); //appel de la fonction
cout << "n=" << n<<endl ;
return 0 ;
}
void sp (int *  nbre)       //Ajout de la référence dans la fonction
{   cout << "----------"<<endl ;
   cout << "nbre=" << *nbre << endl;
    *nbre = *nbre + 2;
   cout << "nbre=" <<* nbre<<endl ;
   cout << "----------"<<endl ;
}
```

Affichage de l'exécution:

```
n=3
----------
nbre=3
nbre=5
----------
n=5

Process returned 0 (0x0)   execution time : 1.674 s
Press any key to continue.
```

III.5 Fonctions *inline* [4b]

Parfois le temps d'exécution d'une fonction est petit comparé au temps nécessaire pour appeler la fonction. Le mot clé ***inline*** informe le compilateur qu'un appel à cette fonction peut être remplacé par le corps de la fonction.

Syntaxe : **inline** type fonct(liste_des_arguments){...}

Exemple

```
inline int max(int a, int b)
{
  return (a < b) ? b : a;
}
void main()
{  int m = max(134, 876);
  cout<< "m=" << m<<endl ;
}
```

- Les fonctions "**inline**" permettent de gagner au niveau de temps d'exécution, mais augmente la taille des programmes en mémoire.
- Contrairement aux macros dans C, les fonctions "**inline**" évitent les effets de bord (dans une macro.
- Les fonctions "**inline**" doivent être définies dans des fichiers d'entête (.h) qui seront inclus dans des fichiers source (.cpp), pour que le compilateur puisse faire l'expansion.