Séance N°5

Fonctions

Une fonction est un ensemble d'instructions qui utilisent des *paramètres* ou variables d'*entrées*, encore appelés *arguments*. Le langage C++ propose de nombreuses fonctions *prédéfinies* pour calculer, par exemple : *sin* calcule le sinus d'un angle ou *sqrt* la racine carrée d'une valeur positive. Ces fonctions prédéfinies (ou *standard*) se trouvent dans des bibliothèques qu'il faut inclure au début du programme pour pouvoir les utiliser (comme *cmath*). Il est aussi possible de construire ses propres fonctions, dites fonctions « *utilisateur* ». Une fonction doit toujours être déclarée et définie avant d'être utilisée dans le programme principal *main()*.

Syntaxe

- si la fonction ne retourne **aucune variable**

void *nom_fonction* (liste des paramètres précédés de leur type séparés par des virgules)

- si la fonction retourne **une variable** (une et une seule)

type_sortie *nomfonction* (liste des paramètres précédés de leur type séparés par des virgules)

La valeur de sortie est délivrée par l'instruction return dans le corps de la fonction.

Ex 1 – Exemple à tester :

```
#include <iostream>
#include <cmath>

using namespace std;

// Fonction qui ne retourne aucune valeur
void affiche_complexe (double reel, double imag)
{
    cout <<"le nombre complexe est : " << reel <<"+ i*" << imag << endl ;
}

// Fonction qui retourne une valeur double
double module (double reel, double imag)
{
    return sqrt(reel*reel+imag*imag) ;
}

// Appel des fonctions dans le programme principal
void main()
{
    double x,y ;
    cout <<"saisir la partie reelle: "; cin >> x ;
    cout<<endl<<"saisir la partie imaginaire: "; cin >> y ;
    affiche_complexe (x, y) ; // Appel de la fonction
    cout<<endl<<"le module est : " << module (x,y) << endl ;
}
```

Question : que signifie le type *void* ?

Passage des paramètres

Une fonction ne peut pas retourner plus d'une valeur. Avant de créer une fonction, il faut bien définir quels sont ses **paramètres** et leurs **types**. Les paramètres sont transmis, **en entrée**, par le programme qui appelle la fonction. Les paramètres d'entrée et de sortie

peuvent être de tout type, y compris des tableaux. Dans ce cas, la dimension du tableau doit être également passée en paramètre si on veut pouvoir faire des boucles de traitement.

Ex 2 – Testez le programme suivant :

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

// Définition d'une fonction de calcul de la moyenne dans un tableau d'entiers
double  moyenne (int  dim, int *tab)
{
      double som = 0;
      for (int  i=0; i<dim; i++)
      {
          som = som+tab[i];
      }
      return som/dim;
}

void main()
{
    const int n = 7;
    int t[n];      // Tableau automatique de n entiers de nom t

    for (int i=0; i< n; i++)
    {
      cout <<"saisir une valeur : "<<endl;
      cin>>t[i] ;
    }

    cout<< endl<<"moyenne = "<< moyenne(n,t)<< endl;
}
```

Rappel : la variable correspondant au nom du tableau, passée en paramètre à la fonction, est un pointeur sur la première adresse de stockage : *t=&t[0]*.

Ex 3 – Cas d'un tableau dynamique (modifier le programme précédent) :

```
// La fonction qui calcule la moyenne d'un tableau est la même que précédemment
// Appel de la fonction dans le programme principal
void  main()
{
    int  n;
    int * t;
    cout<<"Combien d elements ? " ;
    cin>>n;
    t=new int [n];

    if (t)
    {
      for (int  i=0; i<n; i++)
      {
            cout<<endl<<"entrer une valeur : " ;
            cin>>t[i];
      }

      cout<< endl<<"moyenne = "<< moyenne(n,t);
      delete [] t;
    }
}
```

Question: Modifier le code pour calculer la moyenne et l'écart-type d'un tableau de *double*.

Différence entre passage par valeur et par adresse

Lorsqu'une variable est passée en paramètre à une fonction, c'est la **valeur** de la variable qui est transmise à la fonction, la variable originale ne peut donc pas être modifiée dans la fonction mais seulement utilisée. C'est ce qu'on appelle un **passage par valeur.**

Si on veut qu'une **fonction puisse modifier une variable**, il faut qu'elle accède à son stockage en mémoire. Il faut donc transmettre à la fonction l'**adresse de la variable** en paramètre, en utilisant un pointeur. C'est ce que l'on appelle un **passage par adresse**. Un tableau est toujours passé par adresse, puisqu'un tableau est représenté par l'adresse de la première case.

Ex 4 – Passage par valeur :

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

void permute(int a, int b)
{
      int tmp = a;
      a = b;
      b = tmp;
}

void  main()
{
    int a, b;
    cin >> a;
    cin >> b;
    permute(a, b);
    cout << a << " " << b << endl;
}
```

Ex 5 – Passage par adresse :

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

// en utilisant deux pointeurs a et b

void permute(int *a, int *b)
{
      int tmp = *a;
      *a = *b;
      *b = tmp;
}

void  main()
{
    int a, b;
    cin >> a;
    cin >> b;
    permute(&a, &b);
    cout << a << " " << b << endl;
}
```

Passage par référence

En C++, il existe une troisième façon de passer une variable à une fonction qui n'existe pas en C : il s'agit du **passage par référence**. Une référence en C++, est une variable qui fait référence à une autre variable. Les deux variables sont liées et **si l'une est modifiée alors l'autre l'est aussi**. Une variable référence est déclarée de la manière suivante :

type &nom_référence = nom_variable_référencée.

Ex 6 – Passage par référence:

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

// Passage par référence

void permute(int &a, int &b)
{
      int tmp = a;
      a = b;
      b = tmp;
}

void  main()
{
    int a, b;
    cin >> a;
    cin >> b;
    permute(a, b);
    cout << a << " " << b << endl;
}
```

TRAVAIL PERSONNEL

Pour mettre en application ce que j'ai appris, je fais les exercices suivants :

Exercice 1
Écrire la fonction qui prend comme argument une température en Kelvin et qui renvoie sa valeur en degré Fahrenheit.

Exercice 2
En utilisant la fonction permute ci-dessus, écrire un programme avec une fonction qui trie les éléments d'un tableau et délivre la valeur médiane.

Exercice 3
Faire une fonction qui teste si un entier est premier ou pas. Cette fonction renverra un booléen pour indiquer le résultat du test.

Exercice 4
Modifier cette fonction pour qu'elle renvoie aussi le premier entier diviseur s'il en existe ou 1 sinon.

BILAN PERSONNEL

Ce que j'ai appris aujourd'hui : (à compléter)

Vocabulaire informatique : (à compléter)