



T.P. 3

Initiation au langage C++

DEA SyAM 2000/2001

Par
Philippe Lucidarme : lucidarm@lirmm.fr

Initiation au langage C++

Jusqu'à présent, nous n'avons utilisé l'interface Visual C++ que pour réaliser de la programmation en C. Il été impératif de connaître les fonctions de bases avant d'entamer la programmation en C++. En réalité le C++ est une évolution du langage C. Toutes les fonctions et routines du C se retrouvent en C++, mais le C++ propose la POO (Programmation Orientée Objet) que ne possède pas le C.

La surcharge

Ecrire une fonction « somme » qui réalise la somme de deux nombres entiers et renvoie un entiers. Dans le même programme, écrire une deuxième fonction « somme » qui réalise la somme de deux nombres réels et renvoie l'arrondi à l'entier prêt. (Pour convertir un réel en entier, il suffit de le stoker dans une variable entière :

```
float a=5.67 ;
int b ;
b=a ;           // b a pour valeur 5
```

Maintenant, saisissez le programme principal suivant :

```
void main()
{
    int n1=4;
    int n2=5;
    float r1=6.71;
    float r2=4.95;

    printf("la somme des nombres est : %d \n",somme (n1,n2));
    printf("la somme des nombres est : %d \n",somme (r1,r2));
    getch () ;
}
```

Comme vous pouvez le constater, il est possible de définir deux fonctions homonymes. Selon les paramètres, ce ne sera pas la même fonction qui sera appelée. Cela s'appelle la **surcharge**. Dans l'exemple précédent, les deux fonctions avaient le même contenu, mais les fonctions peuvent être complètement différentes.

Surchargez la fonction printf de tel sorte que lorsque l'on écrit printf (entier) l'entier s'affiche à l'écran. Surcharger une nouvelle fois cette fonction de façon à afficher un réel. Ecrire le programme principal qui permet d'afficher un entier et un réel grâce à la fonction printf.

Définition de classes

Depuis que l'informatique existe, on distingue deux éléments essentiels à la construction des programmes : les données et le code. Bien que fondamentalement différents, ces deux éléments se retrouvent étroitement liés à tous les niveaux. La classes se propose de réunir en une seule structure les données et les fonctions associées.

Le C++ offre trois méthodes de définition de classes :

- struct
- union
- class

Nous laisserons de côté le cas de l'union.

Struct

Voici un petit programme qui va nous permettre d'avoir une première approche :

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

struct compte
{
    int somme;

    void credit (int a)
    {        somme=somme+a;    }

    void debit (int a)
    {        somme=somme-a;    }

    int solde (void)
    {        return (somme);    }

    compte ()
    {        somme=0;    }
};

void main (void)
{
    compte c1;
    compte c2;
    c1.credit (1000);    c2.credit (500);    c1.debit (2500);    c2.debit (250);
    printf ("Solde compte 1 : %d\n",c1.solde() );
    printf ("Solde compte 2 : %d\n",c2.solde() );
}
```

Dans notre exemple, nous venons de définir une structure compte. Elle ne possède qu'une seule variable (somme) et trois fonctions (credit, debit et solde). La fonction compte est appelée automatiquement à l'initialisation de la variable (compte c1 ;)

Définition de la structure :

```
Struct nom_structure
{
Déclaration variables ; (appelées données membres)
Déclaration fonctions ; (appelées méthodes)
Initialisation ; (appelée constructeur)
} ;
```

```
// Programme principal
{
nom_structure nom_variable ;
nom_variable.fonction1 ()
}
```

Remarque : l'initialisation est une fonction (ou méthode) qui porte le même nom que la classe, elle est exécutée à chaque nouvelle déclaration de class. Dans notre exemple elle sera exécutée deux fois : compte c1; compte c2; Celui-ci ne renvoie aucun paramètre (pas même un void)

Cette fonction est appelée **constructeur**.

Créez une structure robot qui possède les variables suivantes : Xpos et Ypos : entier, et Orientation : caractère. Xpos et Ypos représentent la position en X et en Y du robot dans son environnement. L'orientation peut prendre les valeurs suivantes : Nord (N), Est (E), Sud (S), Ouest (O). Le robot est initialement placé en 0,0 et orienté vers le nord.

Cette structure possède quatre méthodes :

- X position () : renvoie la position en X.
- Y position () : renvoie la position en Y.
- Avance (int n) : avance de n cases dans la direction courante.
- Tourne () : tourne de 90° dans le sens des aiguilles d'une montre.

Tester la structure en créant deux robots et en les faisant se déplacer avant d'afficher leur position.

Class

Dans ce dernier programme, remplacez struct par class et exécutez le programme :

Error : cannot access private member declared in class 'robot'

Lors de la déclaration d'une structure, les données membres et les fonctions peuvent être accessibles de n'importe quel endroit du programme. Ors, pour éviter les erreurs, nous ne désirons pas que les fonctions extérieures à la classe puissent accéder aux données membres ce qui risquerait de désorganiser le module en lui affectant des valeurs aberrantes.

Il est possible de choisir si une donnée ou une méthode est accessible depuis une autre fonction n'appartenant pas à la classe :

- Public : accès global
< Déclaration>
- Private : accès réservé à la classe
< Déclaration>

Remaque : class déclare toutes ses méthodes et données en private par défaut.

Dans le programme précédent, laissez la déclaration class et rajoutez un public devant les méthodes. Attention : la déclaration public ou private est toujours suivi de « : ».

Les méthodes

Dans tous les exemples précédents, le corps des méthodes s'inscrivait dans le corps de la définition de la classe. Il faut réserver cette solution aux fonctions qui n'occupent que peu de place. Pour des fonctions de taille plus traditionnelle, on préférera la solution de la définition déportée, qui consiste à déclarer la fonction dans la classe et à définir son corps en dehors et en aval de la définition de la classe.

Reprenons l'exemple des comptes :

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

class compte
{
    int somme;
public :
    void credit (int a);
    void debit (int a);
    int solde (void);
    compte ();
};

void compte::credit (int a)
{
    somme=somme+a;
}

void compte::debit (int a)
{
    somme=somme-a;
}

int compte::solde (void)
{
    return (somme);
}

compte::compte ()
{
    somme=0;
}
```

```

void main (void)
{
    compte c1;
    compte c2;
    c1.credit (1000);
    c2.credit (500);
    c1.debit (2500);
    c2.debit (250);
    printf ("Solde compte 1 : %d\n",c1.solde() );
    printf ("Solde compte 2 : %d\n",c2.solde() );
}

```

Comme vous pouvez le constater, le programme principal n'a absolument pas changé, la déclaration des méthodes se trouve maintenant à l'extérieur de la déclaration de la classe.

Le sigle « : » sépare le nom de la classe et le nom de la méthode, pour toutes les méthodes de toutes les classes :

<type> <nom classe> : <nom fonction> <liste arguments>

De la même façon, modifiez la classe robot, de façon à ce que la déclaration des méthodes se fasse à l'extérieur de la classe.

La surcharge operateur

En C++, on peut non seulement surcharger des fonctions, mais aussi des opérateurs standards tels que « + », « - » et d'autres encore. Cela signifie que nous pouvons modifier le comportement de ces opérateurs de telle sorte qu'ils acceptent d'autres types d'opérandes que ceux prévus initialement et que leur action soit entièrement prévu par le programmeur.

Exécutez l'exemple suivant :

```

#include <stdio.h>
#include <conio.h>

class vecteur
{public :
    int X;
    int Y;
    vecteur(int x,int y);
    vecteur();
    vecteur operator= (vecteur a);
    friend vecteur operator+ (vecteur a,vecteur b);
};

vecteur operator+ (vecteur a,vecteur b)
{
    vecteur tmp;
    tmp.X = a.X+b.X;
    tmp.Y = a.Y+b.Y;
    return tmp;
}

```

```

vecteur vecteur::operator=(vecteur a)
{
    this->X=a.X;
    this->Y=a.Y;
    return *this;
}

vecteur::vecteur (int x,int y)
{
    X=x;
    Y=y;
}

vecteur::vecteur ()
{
}

void main (void)
{
    vecteur a (2,3);
    vecteur b (3,8);
    vecteur c;
    c=a+b;
    printf ("a + b -> x=%d y=%d\n",c.X,c.Y);
    getch ();
}

```

Comme vous pouvez le constater, les opérateurs + et = permettent de réaliser des opérations sur d'autres types que ceux définis initialement.

C'est le mot clé operator qui permet cela.

Remarque : le mot clé this représente l'objet courant, dans notre cas l'objet courant est le résultats de l'opération.

En vous inspirant du programme précédant, réalisez un programme qui permet de calculer la somme et le produit de deux matrices carrés (2x2). Vous devrez créer une classe matrice.