Séance N°5

Classe, Constructeurs, Destructeur

La notion de *classe* est ce qui distingue un langage « *orienté objet* », comme le C++, vis-à-vis des langages de programmation structurée comme le C ou le Pascal. Dans ces derniers, les structures de données et les algorithmes qui les utilisent sont déclarés de façon séparée. En C++, on manipule des *objets* qui contiennent à la fois des *données membres* (ou variables) et les *fonctions membres* (ou méthodes) qui les manipulent : ces objets sont définis comme une *classe*. Cette écriture est destinée à améliorer la modularité des logiciels. De nombreuses bibliothèques de fonctions mathématiques ou graphiques en C++ utilisent des classes d'objets spécifiques.

Exemple à tester :

```
#include <iostream>
using namespace std;
class point
{
// Données membres
int x,y;
// Fonctions membres
public:
point(); // constructeur de la classe (pas de "void")
void deplace(int,int);
void affiche();
};

// Définition des fonctions membres
// Constructeur (initialisation par défaut)
point::point()
{x = 20; y = 10;
cout<< "constructeur appele pour point " <<x<< ", "<<y<<endl ;}
// Fonction deplace
void point::deplace(int dx,int dy)
{x = x+dx; y = y+dy;}
// Fonction affiche
void point::affiche()
{cout<<"point en "<<x<<" "<<y<< endl;}

// Utilisation dans le programme principal
void main()
{
point a,b; // les deux points sont initialisés en 20,10
a.affiche();
a.deplace(17,10);
a.affiche();
b.affiche();
}
```

Commentaires :
«point» est une *classe* constituée

- des données membres *x* et *y* ;
- et des fonctions membres : *initialise()*, *deplace()* et *affiche()*.

Les données *x* et y sont par défaut private, c'est-à-dire accessibles seulement par les fonctions membres : on ne peut pas les utiliser dans le programme principal. On dit que le langage C++ réalise l'*encapsulation* des données.
Les fonctions membres peuvent utiliser et modifier les données membres de façon implicite (sans les passer en paramètres). Toutes les fonctions membres sont déclarées comme public, ce qui permet de les appeler de n'importe où.
La classe est déclarée en début de programme, puis on définit le contenu des fonctions membres.

Dans le programme principal, les objets *a* et *b* appartiennent à la *classe* «point», ce sont des variables de type «point».
On a défini ici un nouveau type de variable, propre à cette application.

Constructeurs

Un *constructeur* est une fonction membre particulière qui a le *même* nom que la classe. Le constructeur est appelé pour fabriquer une « instanciation » de la classe, c'est-à-dire pour réserver un emplacement mémoire pour la variable et initialiser les données membres. Une classe bien construite possède toujours au moins un constructeur. Il est possible de déclarer plusieurs constructeurs en variant les paramètres.

Exemple d'un autre constructeur possible de la classe « point » (compléter le programme précédent et tester) :

```
class point
{
// Données membres
int x,y;
// Fonctions membres
public:
point(); // premier constructeur
point(int,int); // second constructeur
void deplace(int,int);
void affiche();
};

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord; // ici paramètres à passer au constructeur
cout<< "constructeur appele pour point " <<x<< ", "<<y<<endl ;}
```

Le compilateur C++ sait automatiquement différentier des fonctions qui portent le même nom mais qui utilisent d'autres paramètres (comme pour la surcharge des opérateurs).

Destructeur

Le destructeur est une fonction membre systématiquement exécutée «à la fin de la vie » d'un objet. Il est absolument nécessaire dés qu'il y a allocation dynamique pour libérer la mémoire (lorsque des données membres sont des pointeurs). Le destructeur porte le nom de la classe, précédé du symbole « ~ », sans aucun paramètre. Il est unique pour chaque classe.

```
~point(); // destructeur de la classe « point »
```

Constructeur de recopie

Ce constructeur est absolument nécessaire pour instancier et initialiser un objet d'une classe à partir d'un autre, si la classe contient des données dynamiques. En l'absence du constructeur de recopie, le compilateur en crée un automatiquement, qui recopie « bit-à-bit » l'objet source dans l'objet destination. Ceci est suffisant pour les objets simples mais pose des problèmes d'allocation mémoire lorsque la classe contient des données dynamiques.

Exemple:

```
#include <iostream>
using namespace std;
// Cas d'une classe qui contient des données dynamiques
class point
{int *t;
public:
point(int, int);
point( point &); // constructeur par recopie (paramètre passé en référence)
~point(); // destructeur nécessaire si données dynamiques pour libérer la mémoire
void affiche();
};
```

```
// constructeur
point::point( int abs, int ord)
{t=new int[2];
t[0]=abs;
t[1]=ord;
cout<<"constructeur du point("<<t[0] <<","<< t[1]<<")"<< endl;
}
// constructeur par recopie (« this » désigne l'adresse de l'objet invoqué)
point::point( point &p)
{t=new int[2];
t[0]=p.t[0];
t[1]=p.t[1];
cout<<"constructeur par recopie du point " << t[0]<<"," << t[1] <<" d'adresse "<<this<<endl;
}
// fonction affiche
void point::affiche()
{cout<<"affichage du point "<< t[0]<<","<< t[1] <<" d'adresse "<<this<<endl;}
// destructeur
point::~point()
{cout<<"destructeur du point("<<t[0]<<","<< t[1]<<")"<<" d'adresse "<<this <<endl;
delete [] t;
}
void main()
{
point a(1,2);
a.affiche();
point b=a; //appel du constructeur par recopie pour instancier b à partir de l'objet a
b.affiche();
cout<<"fin du main"<<endl;
}
```

TRAVAIL PERSONNEL

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

Exercice 1

On définit une classe *CVect2D* (C pour classe) pour traiter des vecteurs à deux dimensions. Cette classe doit contenir 2 variables membres de type double précision que l'on nommera x et y. Elle doit contenir les fonctions suivantes :

- un constructeur de forme *CVect2D(double, double)* qui affiche qu'il a été appelé et initialise x avec le premier paramètre et y avec le second. Faire en sorte que l'on puisse utiliser le constructeur indifféremment avec 0, 1 ou 2 paramètres (lorsque les paramètres sont omis on initialisera les variables à 0) ;
- un destructeur qui affiche qu'il a été appelé ;
- une fonction *affiche* qui affiche les coordonnées du vecteur.

Écrire un programme principal qui utilise les trois formes de construction (0, 1 et 2 paramètres) et la méthode d'affichage de l'objet CVect2D.

Exercice 2

Reprendre la classe *CVect2D* précédente en y ajoutant les fonctions membres suivantes :

- une fonction *homothétie*, qui prend un coefficient réel double précision d'homothétie à appliquer au vecteur ;
- une fonction *module* qui calcule le module du vecteur ;
- une fonction *identique* qui compare l'objet avec un autre objet de la classe CVect2D passé par référence. Cette fonction doit rendre un booléen qui indiquera si les vecteurs sont identiques.

Écrire un programme qui teste les fonctions ci-dessus. Pour cela construisez un vecteur en indiquant les deux coordonnées. Appliquez-lui une homothétie puis construisez un deuxième vecteur sans spécifier de paramètre puis comparez-le au premier (la comparaison sera faite dans les 2 sens).

Exercice 3

Dans cet exercice, on va créer une classe de conversion de température. Pour cela, la classe, que l'on appellera *CTemperature*, doit posséder une variable membre dont la valeur représente la température qui sera convertie. Arbitrairement cette température sera en

Kelvin et la variable membre s'appellera *Temperature*. Pour que cette classe fonctionne correctement, il faut lui adjoindre 3 fonctions membres pour permettre à l'utilisateur de rentrer la température à convertir dans l'unité qui lui plait (*SaisieK* pour une température en Kelvin, *SaisieC* pour les degrés Celsius et *SaisieF* pour les degrés Fahrenheit). Il faut aussi trois fonctions pour récupérer la température convertie (*TempK*, *TempC* et *TempF*).

Exercice 4

Un thermocouple est un capteur composé de deux fils conducteurs de natures différentes. Lorsque la jonction entre ces deux fils est chauffée, il apparaît une différence de potentiel ou ddp E aux bornes du capteur. Cette différence de potentiel (DDP) est proportionnelle à la température T mesurée, mais pas de façon linéaire. La fonction qui permet de transformer la DDP (en mV) en température (en °C) est un polynôme:

$$T = \sum_{i=0}^{n} c_i E^i$$

Dans le cas d'un thermocouple de *type J* travaillant dans la gamme de température [0,760°C], les coefficients c_i du polynôme sont donnés dans le tableau suivant:

```
Coef[0] = 0.0;
Coef[1] = 1.978425e1;
Coef[2] = -2.001204e-1;
Coef[3] = 1.036969e-2;
Coef[4] = -2.549687e-4;
Coef[5] = 3.585153e-6;
Coef[6] = -5.344285e-8;
Coef[7] = 5.099890e-10;
```

Créer un objet *CThermocouple* qui possède les variables membres suivantes :

- le nombre de coefficients du polynôme (NC)
- le tableau qui stocke les coefficients du polynôme de conversion (Coef).

L'objet *CThermocouple* doit aussi contenir les fonctionnalités suivantes :

- un constructeur permettant de définir le type de thermocouple, c'est-à-dire, le nombre de coefficients ainsi que leurs valeurs.
- une fonction *Temperature* qui retourne un objet *CTemperature* lorsqu'on lui spécifie une différence de potentiel en mV.
- une fonction *ChangeCoefficients* qui permet de redéfinir la valeur des coefficients du polynôme de conversion.

Pour des températures comprises dans l'intervalle [0,760°C], la différence de potentiel mesurée aux bornes du thermocouple se situe dans l'intervalle [0, 42.919 mV].

Ecrire un programme qui demande à l'utilisateur une différence de potentiel en mV, effectue la conversion et affiche finalement la température correspondante en Kelvin, en degrés Celsius et en degrés Fahrenheit.

BILAN PERSONNEL

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

Vocabulaire informatique : (à compléter)