COURS et TP DE LANGAGE C++

Chapitre 10

Programmation orientée objet

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 10

PROGRAMMATION ORIENTE OBJET : NOTION DE CLASSE

I- INTRODUCTION

On attend d'un programme informatique :
- l'exactitude (réponse aux spécifications)
- la robustesse (réaction correcte à une utilisation « hors normes »)
- l'extensibilité (aptitude à l'évolution)
- la réutilisabilité (utilisation de modules)
- la portabilité (support d'une autre implémentation)
- l'efficience (performance en termes de vitesse d'exécution et de consommation mémoire)

Les langages évolués de type C ou PASCAL, reposent sur le principe de la programmation structurée (algorithmes + structures de données)
Le C++ et un langage orienté objet. Un langage orienté objet permet la manipulation de *classes*. Comme on le verra dans ce chapitre, la classe généralise la notion de structure.
Une classe contient des variables (ou « données ») et des fonctions (ou « méthodes ») permettant de manipuler ces variables.
Les langages « orientés objet » ont été développés pour faciliter l'écriture et améliorer la qualité des logiciels en termes de modularité et surtout de réutilisation.
Un langage orienté objet est livré avec une bibliothèque de classes. Le développeur utilise ces classes pour mettre au point ses logiciels.

Rappel sur la notion de prototype de fonction:

En C++, comme en C, on a fréquemment besoin de déclarer des *prototypes* de fonctions.
En particulier, on trouve dans les fichiers d'en-tête (de type *.h), des *prototypes* de fonctions appelées par le programme.
Le *prototype* d'une fonction est constitué du nom de la fonction, du type de la valeur de retour, du type des arguments à passer

Exemples: **void ma_fonction1()**

```
void ma_fonction2(int n, float u)   // prototype « complet »
void ma_fonction2(int, float)       // prototype « réduit »

int ma_fonction3(char *x)           // prototype « complet »
int ma_fonction3(char *)            // prototype « réduit »

int ma_fonction4(int &u)            // prototype « complet »
int ma_fonction4(int &)             // prototype « réduit »
```

On utilise indifféremment, dans les fichiers d'en-tête, le prototype complet ou le prototype réduit. Il est recommandé, pour une meilleure lisibilité, d'utiliser le prototype complet.

II- NOTION DE CLASSE

Exemple (à tester) et exercice X-1 :
(il faut ajuster la temporisation aux performances de la machine utilisée)

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs, int ord);
      void deplace(int abs, int ord);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {x = abs; y = ord;}

    void point::deplace(int dx,int dy)
   {x = x+dx; y = y+dy;}

    void point::affiche()
   {gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";}

    void tempo(int duree)
   {
      float stop ;stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);}
// on utilise un float pour obtenir une temporisation suffisamment longue

    void main()
   {
      point a,b;
      a.initialise(1,4);
      a.affiche();
      tempo(10);
      a.deplace(17,10);
      a.affiche();
      b = a;          // affectation autorisée
      tempo(15);
      clrscr();
      b.affiche();
      getch() ;
   }
```

« point » est une ***classe***. Cette classe est constituée des ***données*** *privées x* et y et des ***fonctions membres*** **publiques (*ou méthodes*) «** initialise », « deplace », « affiche ». On déclare la classe en début de programme (données et prototype des fonctions membres), puis on définit le contenu des fonctions membres.
Les données x et y sont dites ***privées.*** Ceci signifie que l'on ne peut les manipuler qu'au travers des fonctions membres publiques. On dit que le langage C++ réalise ***l'encapsulation des données.***
a et b sont des ***objets*** de classe «point», c'est-à-dire des variables de type «point».
On a défini ici un nouveau type de variable, propre à cet application, comme on le fait en C avec les structures.

Suivant le principe dit de « l'encapsulation des données », la notation **a.x** est interdite à l'extérieur de la classe.

Exercice X-2:

Utiliser la classe « point » précédente. Ecrire une fonction de prototype **void test()** dans laquelle on déclare un point u, on l'initialise, on l'affiche , on le déplace et on l'affiche à nouveau. Le programme principal ne contient que l'appel à **test**.

Exercice X-3:

Ecrire une fonction de prototype **void test(point &u)** (référence) similaire. Ne pas déclarer de point local dans **test**. Déclarer un point local a dans le programme principal et appeler la fonction **test** en passant le paramètre a.

Exercice X-4:

Ecrire une fonction de prototype **point test()** qui retourne un point. Ce point sera initialisé et affiché dans **test** puis déplacé et à nouveau affiché dans le programme principal.

III- NOTION DE CONSTRUCTEUR

Un constructeur est une fonction membre *systématiquement exécutée* lors de la déclaration d'un objet statique, automatique, ou dynamique.

On ne traitera dans ce qui suit que les objets automatiques.

Dans l'exemple de la classe **point**, le constructeur remplace la fonction membre **initialise.**

Exemple (à tester) et exercice X-5:

```
#include <iostream.h> // notion de constructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:
      point();
      // noter le prototype du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point() // initialisation sans paramètre
   {
      x = 20; y = 10;
   }  // grace au constructeur

    void point::deplace(int dx,int dy)
   {
      x = x+dx;
      y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);
      cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop ;stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void main()
   {
      point a,b; // les deux points sont initialisés en 20,10
      a.affiche();
      tempo(10);
      a.deplace(17,10);
      a.affiche();
      tempo(15);
      clrscr();
      b.affiche();
      getch() ;
   }
```

Exemple (à tester) et exercice X-6:

```
#include <iostream.h> // introduction au constructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation avec paramètres
   {  // grâce au constructeur, ici paramètres à passer
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void main()
   {  // les deux points sont initialisés : a en (20,10)  b en (30,20)
      point a(20,10),b(30,20);
      a.affiche(); tempo(10);
      a.deplace(17,10);
      a.affiche(); tempo(15);
      clrscr(); b.affiche();
      getch();
   }
```

Exercice X-7: Reprendre l'exercice X-2, en utilisant la classe de l'exercice X-6

Exercice X-8: Reprendre l'exercice X-3, en utilisant la classe de l'exercice X-6

Exercice X-9: Reprendre l'exercice X-4, en utilisant la classe de l'exercice X-6

IV- NOTION DE DESTRUCTEUR

Le destructeur est une fonction membre *systématiquement exécutée* (si elle existe !) «à la fin de la vie » d'un objet statique, automatique, ou dynamique.
On ne peut pas passer de paramètres au destructeur.
Sa tâche est de libérer la mémoire allouée à l'objet lors de sa création (via le constructeur). Si l'on ne définit pas de destructeur, il en existe un par défaut, non visible dans le code, qui assure la même fonction de libération de la mémoire.

On ne traitera dans ce qui suit que les objets automatiques.

Exemple (à tester) et exercice X-10:

```
#include <iostream.h> // notion de destructeur
#include <conio.h>

    class point
   {
      int x,y;
   public:  point(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
      ~point(); // noter la convention adoptée pour le destructeur
   };

    point::point(int abs,int ord)
    // initialisation à partir de paramètres formels
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    point::~point()
   {
      cout<<"Frapper une touche..."; getch();
      cout<<"destruction du point x ="<< x <<" y="<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u(3,7); u.affiche(); tempo(20);
   }

    void main()
   {
      point a(1,4); a.affiche(); tempo(20);
      test();
      point b(5,10); b.affiche(); getch() ;
   }
```

V- ALLOCATION DYNAMIQUE

Lorsque les membres données d'une classe sont des pointeurs, le constructeur est utilisé pour l'allocation dynamique de mémoire sur ce pointeur.
Pour certaines applications, on n'est pas capable de définir la taille d'un tableau au moment de la compilation (exemple : tableau de mesures). La taille effective est fonction d'un contexte d'éxécution. En conséquence, une allocation dynamique s'impose. Les éléments du tableau seront accessibles via un pointeur qui sera une donnée membre privée.
Le destructeur est utilisé pour libérer la place. Il est obligatoire et doit être pris en charge par le programmeur.

Exemple (à tester) et exercice X-11:

```
#include <iostream.h> // Allocation dynamique de données membres
#include <stdlib.h>
#include <conio.h>

    class calcul
   {
   private :
      int nbval,*val;
   public:
      calcul(int nb,int mul);  // constructeur
      ~calcul();               // destructeur
      void affiche();
   };

    calcul::calcul(int nb,int mul)  //constructeur
   {
      nbval = nb;
      val = new int[nbval];  // réserve de la place en mémoire
      for(int i=0;i<nbval;i++)
      {
         val[i] = i*mul;
      }
   }

    calcul::~calcul()
   {
      delete val; // libération de la place réservée
   }

    void calcul::affiche()
   {
      for(int i=0;i<nbval;i++)
      {
         cout<< val[i] <<" ";
      }
      cout<<"\n";
   }

    void main()
   {
      clrscr();
      calcul suite1(10,4);
      suite1.affiche();
      calcul suite2(6,8);
      suite2.affiche();
      getch() ;
   }
```

VII- CORRIGE DES EXERCICES

Exercice X-2:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u;
      u.initialise(1,4); u.affiche();
      tempo(10);
      u.deplace(17,10);  u.affiche();
   }

    void main()
   {
      test(); getch();
   }
```

Exercice X-3:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int,int);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test(point &u)
   {
      u.initialise(1,4);u.affiche();
      tempo(10);
      u.deplace(17,10);u.affiche();
   }

    void main()
   {
      point a;
      test(a);
      getch() ;
   }
```

Exercice X-4:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private :
      int x,y;
   public:
      void initialise(int abs,int ord);
      void deplace(int dx,int dy);
      void affiche();
   };

    void point::initialise(int abs,int ord)
   {
      x = abs; y = ord;
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    point test()
   {
      point u;
      u.initialise(1,4);u.affiche();
      return u;
   }

    void main()
   {
      point a;
      a = test();
      tempo(10);
      a.deplace(17,10);
      a.affiche(); getch() ;
   }
```

Exercice X-7:

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

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test()
   {
      point u(1,4);
      u.affiche();
      tempo(10);
      u.deplace(17,10);
      u.affiche();
   }

    void main()
   {
      test();
      getch();
   }
```

Exercice X-8:

```
#include <iostream.h>   // les classes
#include <conio.h>

    class point
   {
   private :
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence du type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    void test(point &u)
   {
      u.affiche();
      tempo(10);
      u.deplace(17,10);
      u.affiche();
   }

    void main()
   {
      point a(1,4);
      test(a);
      getch() ;
   }
```

Exercice X-9:

```
#include <iostream.h> // les classes
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point(int abs,int ord);
    // noter l'absence de type de retour du constructeur (pas de "void")
      void deplace(int dx,int dy);
      void affiche();
   };

    point::point(int abs,int ord) // initialisation par des paramètres
   {
      x = abs; y = ord; // grâce au constructeur, ici paramètres à passer
   }

    void point::deplace(int dx,int dy)
   {
      x = x+dx; y = y+dy;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<< x <<"  "<< y <<"\n";
   }

    void tempo(int duree)
   {
      float stop = duree*10000.0;
      for(;stop>0;stop=stop-1.0);
   }

    point test()
   {
      point u(5,6); u.affiche();
      return u;
   }

    void main()
   {
      point a(1,4);
      a.affiche(); tempo(15);
      a = test();  tempo(10);
      a.deplace(17,10); a.affiche();
   }
```