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.
Un langage orienté objet sera 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. Par exemple, dans les fichiers d'en-tête (de type *.h), sont déclarés les *prototypes* des 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.

II- NOTION DE CLASSE

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

```
#include <iostream.h>      // les classes
#include <conio.h>
class point
{
int x,y;
public:  void initialise(int,int);
         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 ;stop = duree*10000.0;for(;stop>0;stop=stop-1.0);}

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

«point» est une ***classe***. Cette classe est constituée des ***données*** *x* et y et des ***fonctions membres*** **(*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. 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.

Exercice II-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 *main* ne contient que l'appel à *test.*

Exercice II-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 *main* et appeler la fonction *test* en passant le paramètre a.

Exercice II-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 *main.*

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 des objets automatiques.

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

Exemple (à tester) et exercice II-5:

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

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

point::point() // initialisation par default
{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 II-6:

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

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

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord;}  // grace 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 ;stop = duree*10000.0;for(;stop>0;stop=stop-1.0);}

void main()
{
```

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

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

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

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

IV- NOTION DE DESTRUCTEUR

Le destructeur est une fonction membre *systématiquement exécutée* «à la fin de la vie » d'un objet statique, automatique, ou dynamique.
On ne peut pas passer de paramètres par le destructeur.

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

Exemple (à tester) et exercice II-10:

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

class point
{
int x,y;
public:  point(int,int);
          void deplace(int,int);
          void affiche();
         ~point();       // noter le type du destructeur
};

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord;}  // grace 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 ;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.
Le destructeur est utilisé pour libérer la place.

Exemple (à tester) et exercice II-11:

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

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

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

calcul::~calcul()
{delete val;}  // abandon de la place reservee
```

```
void calcul::affiche()
{int i;
for(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() ;}
```

VI- EXERCICES RECAPITULATIFS

Exemple (à tester) et exercice II-12:

Cet exemple ne fonctionne qu'en environnement DOS. Il utilise les fonctions graphiques classiques du TURBO C. On crée une classe « losange », les fonctions membres permettent de manipuler ce losange.

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

class losange
{
int x,y,dx,dy,couleur;
public:
losange();
void deplace(int,int,int);
void affiche();
void efface();
};

losange::losange() // constructeur
{x=100;y=100;dx=60;dy=100;couleur=BLUE;}

void losange::deplace(int depx,int depy,int coul)
{x=x+depx;y=y+depy;couleur=coul;}

void losange::affiche()
{int tab[10];
tab[0]=x;tab[1]=y;tab[2]=x+dx/2;tab[3]=y+dy/2;
```

```
tab[4]=x;tab[5]=y+dy;tab[6]=x-dx/2;tab[7]=y+dy/2;
tab[8]=x;tab[9]=y;
setfillstyle(SOLID_FILL,couleur);
fillpoly(5,tab);
}

void losange::efface()
{int tab[10];
tab[0]=x;tab[1]=y;tab[2]=x+dx/2;tab[3]=y+dy/2;
tab[4]=x;tab[5]=y+dy;tab[6]=x-dx/2;tab[7]=y+dy/2;
tab[8]=x;tab[9]=y;
setcolor(getbkcolor());  // pour effacer le contour
setfillstyle(SOLID_FILL,getbkcolor());
fillpoly(5,tab);
}

void init_graph()
{
int gd,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\cplus\\bgi");
setbkcolor(YELLOW);
}

void main()
{
losange l;
init_graph();
l.affiche();
getch();closegraph();
}
```

Exercice II- 13: Modifier le programme principal de sorte de faire clignoter le losange tant que l'utilisateur n'a pas appuyé sur une touche.

Exercice II- 14: Modifier le programme principal de sorte de déplacer le losange d'une position à une autre position, tant que l'utilisateur n'a pas appuyé sur une touche.

VII- CORRIGE DES EXERCICES

Exercice II-2:

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

class point
{
int x,y;
```

```
public:  void initialise(int,int);
         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 ;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 II-3:

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

class point
{
int x,y;
public:  void initialise(int,int);
         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 ;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 II-4:

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

class point
{
int x,y;
public:  void initialise(int,int);
         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 ;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 II-7:

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

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

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord;}  // grace 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 ;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 II-8:

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

class point
{
int x,y;
public:  point(int,int); // noter le type du constructeur (pas de "void")
```

```
        void deplace(int,int);
        void affiche();
};

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord;}  // grace 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 ;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 II-9:

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

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

point::point(int abs,int ord) // initialisation par default
{x = abs; y = ord;}  // grace 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 ;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();}
```

Exercice II-13:

```
void main()
{
losange l;
init_graph();
while(!kbhit())
{
l.affiche(); delay(500);
l.efface(); delay(500);
}
getch();closegraph();
}
```

Exercice II-14:

```
void main()
{
losange l;
init_graph();
while(!kbhit())
{
l.affiche(); delay(500);l.efface();
l.deplace(150,150,RED);
l.affiche();delay(500);l.efface();
l.deplace(-150,-150,BLUE);
}
getch();closegraph();
}
```