

# Orientée Objet avec le langage C++

## Travaux Pratiques (constructeur et destructeur)

### Exercice N°1

1. Réaliser une classe **Point** permettant de manipuler un point d'un plan. On prévoira :
  - Un **constructeur** recevant en arguments les coordonnées (réels) d'un point ;
  - Une méthode **deplace** effectuant une translation définie par ses deux arguments ;
  - Une méthode **affiche** se contentant d'afficher les coordonnées cartésiennes du point et le nombre d'objets de type Point.

On écrira séparément :

- Un **fichier en-tête** constituant la déclaration de la classe ;
  - Un **fichier source** correspondant à sa définition ;
  - Un **fichier main** déclarant des points, les affichant, les déplaçant et les réaffichant.
2. Adapter le projet pour qu'il permette d'accéder aux coordonnées d'un point tout en respectant le principe d'encapsulation.
  3. Ajouter à la classe Point de nouvelles méthodes :
    - **homothetie** qui effectue une homothétie dont le rapport est fourni en argument ;
    - **rotation** qui effectue une rotation dont l'angle est fourni en argument ;
    - **rho** et **theta** qui fournissent les coordonnées polaires du point.
  4. Modifier la classe Point de manière que les attributs soient les coordonnées polaires d'un point et non plus ses coordonnées cartésiennes. On évitera de modifier la déclaration des méthodes, de sorte que l'interface de la classe ne change pas.
  5. **N.B** : Un attribut déclaré comme statique se comporte comme une variable globale, c'est-à-dire accessible partout dans le code. Exemple : dans un fichier Etudiant.cpp, on peut écrire :  
int Etudiant :: nbr = 0 ; L'attribut nbr sera donc accessible dans tout le projet en question.

### Exercice N°2

1. Créer une classe **Set\_char** permettant de manipuler des ensembles de caractères. On devra pouvoir réaliser sur un tel ensemble les opérations classiques suivantes : lui ajouter un nouvel élément, connaître son cardinal, savoir si un caractère donné lui appartient.  
Utiliser la classe pour déterminer le nombre de caractères différents contenus dans un mot lu en donnée et tester si des caractères donnés appartiennent au mot lu.
2. Modifier la classe **Set\_char** de manière à disposer de ce que l'on nomme un itérateur sur les différents éléments de l'ensemble. Il s'agit d'un mécanisme permettant d'accéder séquentiellement aux différents éléments. On prévoira trois nouvelles méthodes :

- **init** qui initialise le processus d'exploration ;
- **prochain** qui fournit la valeur de l'élément suivant lorsqu'il existe ;
- **existe** qui précise s'il existe encore un élément non exploré.

On complètera alors le programme de manière qu'il affiche les différents caractères contenus dans le mot fourni en donnée.

## Solutions :

Exercice N°1 :

### 1. Fichier source (nommé point.h)

```
#ifndef POINT1_H
#define POINT1_H
/* fichier POINT1.H */
/* déclaration de la classe point */
class point
{
float x, y ; // coordonnées (cartésiennes) du point
public :
point (float, float) ; // constructeur
void deplace (float, float) ; // déplacement
void affiche () ; // affichage
} ;
#endif
```

#### Fichier source contenant la définition de la classe

```
/* définition de la classe point */
#include "point1.h"
#include <iostream>
using namespace std ;
point::point (float abs, float ord)
{ x = abs ; y = ord ;
}
void point::deplace (float dx, float dy)
{ x = x + dx ; y = + dy ;
}
void point::affiche ()
{ cout << "Mes coordonnées cartésiennes sont " << x << " " << y << "\n" ;
}
```

#### Fichier Main

```
#include <iostream>
using namespace std ;
#include "point1.h"
main ()
{
```

```

point p (1.25, 2.5) ; // construction d'un point de coordonnées 1.25 2.5
p.affiche () ; // affichage de ce point
p.deplace (2.1, 3.4) ; // déplacement de ce point
p.affiche () ; // nouvel affichage
}

```

- Il suffit d'introduire deux nouvelles fonctions membre abscisse et ordonnee et de supprimer la fonction affiche.  
La nouvelle déclaration de la classe est alors :

```

/* fichier POINT2.H */
/* déclaration de la classe point */
class point
{
float x, y ; // coordonnées (cartésiennes) du point
public :
point (float, float) ; // constructeur
void deplace (float, float) ; // déplacement
float abscisse () ; // abscisse du point
float ordonnee () ; // ordonnée du point
} ;

```

#### **Fichier source contenant la nouvelle définition de la classe**

```

/* définition de la classe point */
#include "point2.h"
#include <iostream>
using namespace std ;
point::point (float abs, float ord)
{ x = abs ; y = ord ;
}
void point::deplace (float dx, float dy)
{ x = x + dx ; y = + dy ;
}
float point::abscisse ()
{ return x ;
}
float point::ordonnee ()
{ return y ;
}

```

#### **Fichier Main**

```

/* exemple d'utilisation de la classe point */
#include "point2.h"
#include <iostream>
using namespace std ;
main ()
{
point p (1.25, 2.5) ; // construction
// affichage
cout << "Coordonnées cartésiennes : " << p.abscisse () << " " << p.ordonnee () << "\n" ;
p.deplace (2.1, 3.4) ; // déplacement
// affichage
cout << "Coordonnées cartésiennes : " << p.abscisse () << " " << p.ordonnee () << "\n" ;
}

```

**ESTL –GI**

**Professeur: A.ENNACIRI**

**2019/2020**

```
}
```

3.

La déclaration de la nouvelle classe point découle directement de l'énoncé :

```
/* fichier POINT3.H */  
/* déclaration de la classe point */  
class point  
{ float x, y ; // coordonnées (cartésiennes) du point  
public :  
point (float, float) ; // constructeur  
void deplace (float, float) ; // déplacement  
void homothetie (float) ; // homothétie  
void rotation (float) ; // rotation  
float abscisse () ; // abscisse du point  
float ordonnee () ; // ordonnée du point  
float rho () ; // rayon vecteur  
float theta () ; // angle  
};
```

*Sa définition mérite quelques remarques. En effet, si homothetie ne présente aucune difficulté, la fonction membre rotation nécessite quant à elle une transformation intermédiaire des coordonnées cartésiennes du point en coordonnées polaires. De même, la fonction membre rho doit calculer le rayon vecteur d'un point dont on connaît les coordonnées cartésiennes tandis que la fonction membre theta doit calculer l'angle d'un point dont on connaît les coordonnées cartésiennes. Le calcul de rayon vecteur étant simple, nous l'avons laissé figurer dans les deux fonctions concernées (rotation et rho). En revanche, le calcul d'angle a été réalisé par ce que nous nommons une « fonction de service », c'est-à-dire une fonction qui n'a d'intérêt que dans la définition de la classe elle-même. Ici, il s'agit d'une fonction indépendante mais, bien entendu, on peut prévoir des fonctions de service sous forme de fonctions membre (elles seront alors généralement privées). Voici finalement la définition de notre classe point :*

#### **Fichier source contenant la nouvelle définition de la classe**

```
/ déclarations de service/  
#include "point3.h"  
#include <cmath> // pour sqrt et atan  
#include <iostream>  
using namespace std ;  
const float pi = 3.141592653 ; // valeur de pi  
float angle (float, float) ; // fonction de service (non membre)  
/***** définition des fonctions membre *****/  
point::point (float abs, float ord)  
{ x = abs ; y = ord ;  
}  
void point::deplace (float dx, float dy)  
{ x += dx ; y += dy ;  
}  
void point::homothetie (float hm)  
{ x *= hm ; y *= hm ;  
}
```

```

void point::rotation (float th)
{ float r = sqrt (x*x + y*y) ; // passage en
float t = angle (x, y) ; // coordonnées polaires
t += th ; // rotation th
x = r * cos (t) ; // retour en
y = r * sin (t) ; // coordonnées cartésiennes
}
float point::abscisse ()
{ return x ;
}
float point::ordonnee ()
{ return y ;
}
float point::rho ()
{ return sqrt (x*x + y*y) ;
}
float point::theta ()
{ return angle (x, y) ;
}
/ définition des fonctions de service /
/* fonction de calcul de l'angle correspondant aux coordonnées */
/* cartésiennes fournies en argument */
/* On choisit une détermination entre -pi et +pi (0 si x=0) */
float angle (float x, float y)
{ float a = x ? atan (y/x) : 0 ;
if (y<0) if (x>=0) return a + pi ;
else return a - pi ;
return a ;
}

```

La déclaration de la nouvelle classe découle directement de l'énoncé :

```

/* fichier POINT4.H : déclaration de la classe point */
class point
{ float r, t ; // coordonnées (polaires) du point
public :
point (float, float) ; // constructeur
void deplace (float, float) ; // déplacement
void homothetie (float) ; // homothétie
void rotation (float) ; // rotation
float abscisse () ; // abscisse du point
float ordonnee () ; // ordonnée du point
float rho () ; // rayon vecteur
float theta () ; // angle
} ;

```

*En ce qui concerne sa définition, il est maintenant nécessaire de remarquer que :*

- *le constructeur reçoit toujours en argument les coordonnées cartésiennes d'un point ; il doit donc opérer les transformations appropriées ;*
- *la fonction deplace reçoit un déplacement exprimé en coordonnées cartésiennes ; il faut donc tout d'abord déterminer les coordonnées cartésiennes du point après déplacement, avant de repasser en coordonnées polaires.*

*En revanche, les fonctions homothétie et rotation s'expriment très simplement.*

*Voici la définition de notre nouvelle classe (nous avons fait appel à la même fonction de service angle que dans la question précédente) :*

```
#include "point4.h"
#include <cmath> // pour cos, sin, sqrt et atan
#include <iostream>
using namespace std ;
const int pi = 3.141592635 ; // valeur de pi

/définition des fonctions de service /
/* fonction de calcul de l'angle correspondant aux coordonnées */
/* cartésiennes fournies en argument */
/* On choisit une détermination entre -pi et +pi (0 si x=0) */
float angle (float x, float y)
{ float a = x ? atan (y/x) : 0 ;
  if (y<0) if (x>=0) return a + pi ;
  else return a - pi ;
  return a ;
}
/ définition des fonctions membre /
point::point (float abs, float ord)
{ r = sqrt (abs*abs + ord*ord) ;
  t = atan (ord/abs) ;
}
void point::deplace (float dx, float dy)
{ float x = r * cos (t) + dx ; // nouvelle abscisse
  float y = r * sin (t) + dy ; // nouvelle ordonnée
  r = sqrt (x*x + y*y) ;
  t = angle (x, y) ;
}
void point::homothetie (float hm)
{ r *= hm ;
}
void point::rotation (float th)
{ t += th ;
}
float point::abscisse ()
{ return r * cos (t) ;
}
float point::ordonnee ()
{ return r * sin (t) ;
}
float point::rho ()
{ return r ;
}
float point::theta ()
{ return t ;
}
```

## Exercice N°2 :

Le reste de la déclaration de la classe découle de l'énoncé.

```
/* fichier SETCHAR1.H */
/* déclaration de la classe set_char */
#define N_CAR_MAX 256 // on pourrait utiliser UCHAR_MAX défini
// dans <limits.h>
class set_char
{
    unsigned char ens [N_CAR_MAX] ;
    // tableau des indicateurs (présent/absent)
    // pour chacun des caractères possibles
public :
    set_char () ; // constructeur
    void ajoute (unsigned char) ; // ajout d'un élément
    int appartient (unsigned char) ; // appartenance d'un élément
    int cardinal () ; // cardinal de l'ensemble
} ;
```

### Fichier source contenant la définition de la classe

```
/* définition de la classe set_char */
#include "setchar1.h"
set_char::set_char ()
{ int i ;
  for (i=0 ; i<N_CAR_MAX ; i++) ens[i] = 0 ;
}
void set_char::ajoute (unsigned char c)
{ ens[c] = 1 ;
}
int set_char::appartient (unsigned char c)
{ return ens[c] ;
}
int set_char::cardinal ()
{ int i, n ;
  for (i=0, n=0 ; i<N_CAR_MAX ; i++) if (ens[i]) n++ ;
  return n ;
}
```

Il en va de même pour le programme d'utilisation :

```
/* utilisation de la classe set_char */
#include <cstring>
#include "setchar1.h"
#include <iostream>
using namespace std ;
main()
{ set_char ens ;
  char mot [81] ;
  cout << "donnez un mot " ;
  cin >> mot ;
```

```

int i ;
for (i=0 ; i<strlen(mot) ; i++) ens.ajoute (mot[i]) ;
cout << "il contient " << ens.cardinal () << " caractères différents" ;
if (ens.appartient('e')) cout << "le caractère e est présent\n" ;
else
cout << "le caractère e n'est pas présent\n" ;
}

```

N.B :Si l'on avait déclaré de type char les arguments de ajoute et appartient, on aurait alors pu aboutir soit au type unsigned char, soit au type signed char, selon l'environnement utilisé.

Dans le dernier cas, on aurait couru le risque de transmettre à l'une des fonctions membre citées une valeur négative, et partant d'accéder à l'extérieur du tableau ens.

*Remarque : Le tableau ens [N\_CHAR\_MAX] occupe un octet par caractère ; chacun de ces octets ne prend que l'une des valeurs 0 ou 1 ; on pourrait économiser de l'espace mémoire en prévoyant seulement 1 bit par caractère. Les fonctions membre y perdraient toutefois en simplicité, ainsi qu'en vitesse. Bien entendu, beaucoup d'autres implémentations sont possibles ; c'est ainsi, par exemple, que l'on pourrait fournir au constructeur un nombre maximal d'éléments, et allouer dynamiquement l'emplacement mémoire correspondant ; toutefois, là encore, on perdrait le bénéfice de la correspondance immédiate entre un caractère et la position de son indicateur. Notez toutefois que ce sera la seule possibilité réaliste lorsqu'il s'agira de représenter des ensembles dans lesquels le nombre maximal d'éléments sera très grand.*

2. Compte tenu de l'implémentation de notre classe, la gestion du mécanisme d'itération nécessite l'emploi d'un pointeur (que nous nommerons courant) sur un élément du tableau ens. Nous conviendrons que courant désigne le premier élément de ens non encore traité dans l'itération, c'est-à-dire non encore renvoyé par la fonction membre suivant (nous aurions pu adopter la convention contraire, à savoir que courant désigne le dernier élément traité). En outre, pour faciliter la reconnaissance de la fin de l'itération, nous utiliserons un membre donnée supplémentaire (fin) valant 0 dans les cas usuels, et 1 lorsqu'aucun élément ne sera disponible (pour suivant). Le rôle de la fonction init sera donc de faire pointer courant sur la première valeur non nulle de ens s'il en existe une ; dans le cas contraire, fin sera placé à 1. La fonction suivant fournira en retour l'élément pointé par courant lorsqu'il existe (fin non nul) ou la valeur 0 dans le cas contraire (il s'agit là d'une convention destinée à protéger l'utilisateur ayant appelé cette fonction, alors qu'aucun élément n'était plus disponible). Dans le premier cas, suivant recherchera le prochain élément de l'ensemble (en modifiant la valeur de fin lorsqu'un tel élément n'existe pas). Notez bien qu'ici la fonction suivant doit renvoyer non pas le prochain élément, mais l'élément courant. Enfin, la fonction existe se contentera de renvoyer la valeur de fin puisque cette dernière indique l'existence ou l'inexistence d'un élément courant.

```

/* fichier SETCHAR2.H */
/* déclaration de la classe set_char */
#define N_CAR_MAX 256 // on pourrait utiliser UCHAR_MAX défini
    // dans <climits>
class set_char
{
    unsigned char ens [N_CAR_MAX] ;
    // tableau des indicateurs (présent/absent)
    // pour chacun des caractères possibles
    int courant ; // position courante dans le tableau ens
    int fin ; // indique si fin atteinte

```

```

public :
set_char () ; // constructeur
void ajoute (unsigned char) ; // ajout d'un élément
int appartient (unsigned char) ; // appartenance d'un élément
int cardinal () ; // cardinal de l'ensemble
void init () ; // initialisation itération
unsigned char suivant () ; // caractère suivant
int existe () ;
};

```

Voici la définition des trois nouvelles fonctions membre init, suivant et existe :

```

void set_char::init ()
{ courant=0 ; fin = 0 ;
while ( (++courant<N_CAR_MAX) && (!ens[courant]) ) ;
// si la fin de ens est atteinte, courant vaut N_CAR_MAX
if (courant>=N_CAR_MAX) fin = 1 ;
}

unsigned char set_char::suivant ()
{ if (fin) return 0 ; // au cas où on serait déjà en fin de ens
unsigned char c = courant ; // conservation du caractère courant
// et recherche du suivant s'il existe
while ( (++courant<N_CAR_MAX) && (!ens[courant]) ) ;
// si la fin de ens est atteinte, courant vaut N_CAR_MAX
if (courant>=N_CAR_MAX) fin = 1 ; // s'il n'y a plus de caractère
return c ;
}

int set_char::existe ()
{ return (!fin) ;
}

```

Voici enfin l'adaptation du programme d'utilisation :

```

#include <cstring>
#include "setchar2.h"
#include <iostream>
using namespace std ;

```

main()

```
{ set_char ens ;
char mot [81] ;
cout << "donnez un mot " ;
cin >> mot ;
int i ;
for (i=0 ; i<strlen(mot) ; i++) ens.ajoute (mot[i]) ;
cout << "il contient " << ens.cardinal () << " caractères différents"
<< " qui sont :\n" ;
ens.init() ; // init itération sur les caractères de l'ensemble
while (ens.existe())
cout << ens.suivant () ;
}
```