TP 1 - Exercices

Pour chaque exercice, créer un nouveau projet. Commenter et conserver les fichiers modifiés. Rédiger les réponses dans un fichier texte.

Exercice 1 : Rappels...

1-a Ecrire un programme (en C++) convivial demandant à l'utilisateur 2 nombres entiers. Le programme affiche ensuite : la somme, la différence, le produit, le quotient, le reste de la division entière entre ces 2 nombres.

1-b Modifier ce programme en demandant à l'utilisateur quelle opération il veut réaliser.

Exemple d'exécution :

```
Entrer le premier nombre : 12
Entrer le deuxième nombre : 2
Opération à effectuer ? ( +, -, *, /, %) : *
Le résultat de la multiplication de 12 par 2 est 24.
```

Exercice 2 : Tableau et Fonctions

2-a Ecrire un programme permettant de saisir le contenu d'un tableau de nombres réels. Le programme affichera ensuite les valeurs entrées. Le nombre de valeurs à entrer (nombre d'éléments) est demandé à l'utilisateur (utiliser pour l'instant une allocation statique de la mémoire : `double tab[100]`).

2-b Créer la procédure **`AfficherTableau`** qui permet d'afficher tous les éléments d'un tableau (le nombre d'éléments du tableau doit nécessairement être passé en paramètre). Modifier le programme afin d'utiliser cette fonction.

2-c Créer la fonction **`SaisirNbElements`** qui demande à l'utilisateur le nombre d'éléments à entrer. Utiliser cette fonction dans le programme précédent.

2-d Ajouter au programme précédent la procédure **`SaisirTableau`**.

2-e Modifier le programme afin d'utiliser une allocation dynamique pour le tableau : le nombre d'éléments à entrer sera alors égal à la taille du tableau.

Exemple d'allocation dynamique :

```
int nb ;
double *tab = NULL ;// declaration + affectation du pointeur
cout << "Nombre d'elements ? ";
cin >> nb ;
tab = new double[nb] ;  // allocation de l'espace memoire
… // utilisation du tableau
…
```

```
delete[] tab ;  // libération de l'espace mémoire
```

2-f Faire la fonction **`AdditionneTableaux`**, permettant de faire la somme terme à terme de deux tableau de même taille. Tester cette fonction avec 2 tableaux (tab1 + tab2) puis avec 3 tableaux (le but est de faire la somme termes à termes de trois tableaux).

Exercice 3 : Premières classes

Voici le diagramme de la classe **`Point`** ainsi que sa définition (.h) et son implémentation (.cxx). Cette classe permet de représenter et de manipuler un point du plan. Le but sera de manipuler des formes géométriques basées sur cette classe.

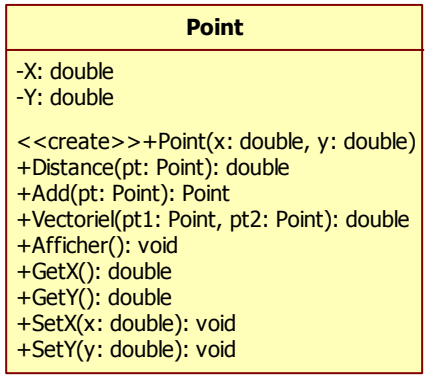

Diagramme de la classe Point

```
#ifndef _Point_h_
#define _Point_h_



class Point
{
private:
    double X;
    double Y;

 public:

 Point(double x = 0, double y = 0);

 Point Add(Point pt);

 void Afficher();

 double GetX();
 double GetY();

 void SetX(double x);
 void SetY(double y);

};


#endif
```

TP1_Ex3_Point.h

```
#include <iostream.h>
#include <math.h>
#include "TP1_Ex3_Point.h"

Point::Point(double x, double y)
{
 X=x;
 Y=y;
}

Point Point::Add(Point pt)
{
 Point result(pt.X + X, pt.Y + Y);
 return result;
}

void Point::Afficher()
{
 cout<< "(" << X << " ," << Y << " )";
}

double Point::GetX()
{ return X; }

double Point::GetY()
{  return Y; }

void Point::SetX(double x)
{  X = x; }

void Point::SetY(double y)
{  Y = y; }
```

TP1_Ex3_Point.cxx

3-a Analyser le programme suivant (comprendre chaque ligne du code et les résultats obtenus). Les 3 fichiers (ce programme (TP1_Ex3_main.cxx) et les deux fichiers TP1_Ex3_Point.h et TP1_Ex3_Point.cxx) sont sur le réseau (*moodle → GE → support de cours → Informatique → Informatique 4*). Déboguer et au besoin modifier ces fichiers pour comprendre (constructeurs, pointeurs, appels des fonctions membres, …).

```
#include <iostream.h>
#include "TP1_Ex3_Point.h"

int main( void )
{
cout << " #### TP1 exo 3 #### " << endl;

Point pt1(1,1);
Point pt2;
Point * _pt;

pt2 = Point(3,1);
_pt = new Point();

cout << "Position de pt1 : ";
pt1.Afficher();
pt1.SetX( 2 ); pt1.SetY( 3 );
pt1.Afficher();
cout << endl;

cout << "Position de pt2 : ";
pt2.Afficher();
cout << endl;

*_pt = pt1.Add(pt2);

cout << "Position de pt3 : ";
_pt->Afficher();
cout << endl;

delete _pt;

cout << "Appuyer sur Entree pour continuer" << endl;
cin.get();
return 0;
}
```

Programme à analyser

3-b Modifier la classe **`Point`** pour ajouter la fonction membre « **`Distance`** » qui calcule la distance entre deux points : c'est-à-dire entre le point courant et un point passé en paramètre à la fonction. Tester cette méthode dans le programme.

3-c Créer une classe **`Triangle`**, permettant de définir un triangle dans le plan à partir de 3 objets **`Point`**. Donner à cette classe les fonctionnalités suivantes :

- Les fonctions d'accès en lecture et écriture aux champs (les 3 points).
- Affichage : pour afficher les coordonnées des 3 points.
- Perimetre : qui retourne le périmètre du triangle.
- Translation : qui translate le triangle par un vecteur (utiliser un objet Point pour le vecteur).
- Aire : qui retourne l'aire du triangle (formule de Héron ou ½ périmètre).

$$Aire = \sqrt{p.(p-a).(p-b).(p-c)} \quad avec\ p = (a+b+c)/2$$

Avec a, b et c les longueurs des cotés

3-d Concevoir (ne pas implémenter les fonctions) une classe **`Quadri`** reprenant toutes les fonctionnalités de la classe triangle, mais pour un quadrilatère… Un quadrilatère sera défini par 4 points (les cas « papillon » seront interdits).

3-e Concevoir (ne pas implémenter les fonctions) une classe **`Cercle`** reprenant toutes les fonctionnalités de la classe triangle, mais pour un cercle… Un cercle sera défini par un point « centre » et un rayon.

3-f L'intérêt principal de la programmation orientée objet est la mutualisation du code (mise en commun et regroupement dans un seul objet des fonctions utilisées par d'autres objets). Identifier dans les classes `Triangle`, `Carre` et `Cercle` les points communs : champs et fonctions membres. Comment modifier ces classes pour rendre les fonctions membres semblables (même algorithme) ? Existe-t-il une (ou plusieurs) fonction(s) membre des trois classes non compatible avec votre schéma ?

3-g Voici le diagramme de classes proposé pour mutualiser le code.

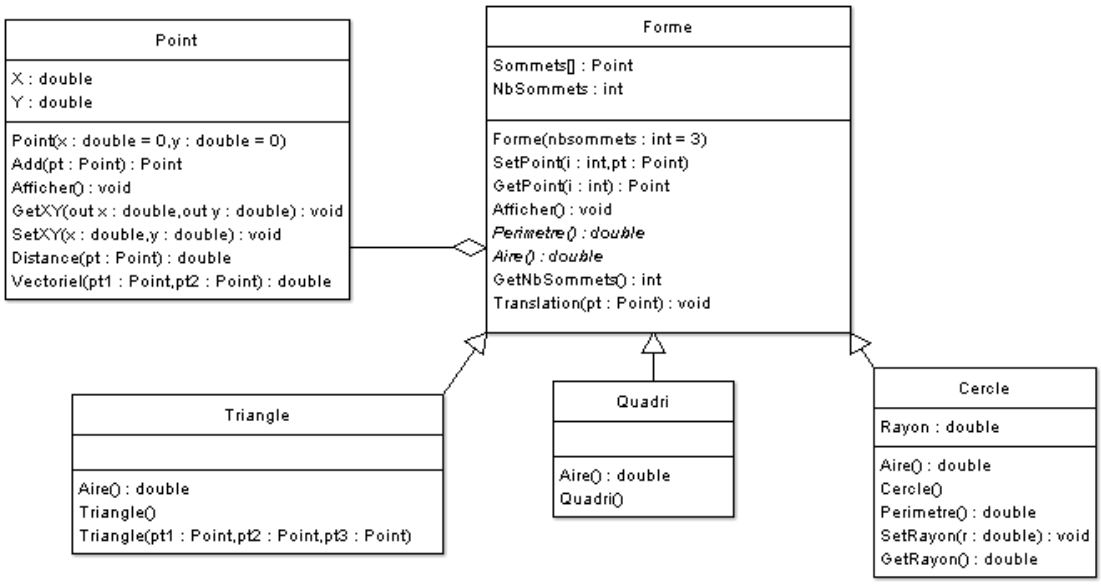

Diagramme de l'application

Ecrire les nouvelles classes **`Quadri`** et **`Cercle`**. Les classes **`Point`**, **`Forme`** et **`Triangle`** sont données, ainsi qu'un exemple d'utilisation.

Remarques pour le calcul de l'aire des quadrilatères : (on ne considère pas les quadrilatères papillons, ni triangles...)

- Un quadrilatère peut se décomposer en 2 triangles de 2 manières différentes.
- Dans le cas des quadrilatères **concaves**, un seul des 2 découpages conduit au bon calcul de l'aire. Pour trouver le bon découpage il faut trouver les 2 points non connexes dont les 2 angles pour passer d'un point à l'autre sont de signes opposés. Exemple, dans la figure ci-dessous les 2 découpages possibles, et les triangles obtenus, sont :
  - segment de coupe [AC], triangles (C, A, D) et (C, A, B) : mauvais découpage,
  - segment de coupe [BD], triangles (B, D, A) et (B, D, C) : bon découpage.

Les angles pour le découpage suivant [AC] sont ADC et ABC : ils sont de même signe
➔ mauvais découpage.

Les angles pour le découpage suivant [BD] sont DAB et DCB : ils sont de signes opposés
➔ bon découpage.