```
/**********************
Cercle.h
**********************/
#pragma once
#define _USE_MATH_DEFINES
#include <math.h>
#include "Point2D.h"
#include "Figure.h"

// #define PI 3.14159265358979323846

class Point2D;
class Figure;

class Cercle : public Figure
{
private :
        double rayon;
        Point2D centre;

public:
        Cercle(Point2D centre, double rayon);
        double getPerimetre();
        double getSurface();
};



/**********************
Commande.h
**********************/
#pragma once
#include <vector>
#include <string>
#include "Figure.h"

using namespace std;

class Figure;

class Commande
{
private:
        bool commandeTerminee;
        double prixMetreDecoupe , prixMetreCarreMatiere ;
        string idCommande;
        vector<Figure*> lesFigures;

public:
        Commande(string identifiantCommande , double lePrixMetreDecoupe , double lePrixMetreCarreMatiere);
        string getIdCommande() { return idCommande; }
        void ajouterNouvelleFigure(Figure *laFigure);
        void cloturerCommande();

        double getPrix() ;

};



/**********************
Figure.h
**********************/
#pragma once

class Figure
{
public:
        virtual double getPerimetre() = 0;
        virtual double getSurface() = 0;
};

/**********************
Point2D.h
**********************/
// Cette classe n'est pas à modifier
#pragma once

class Point2D
{
private:
        double x , y ;

public:
        Point2D(double x=0 , double y=0);
        double getX();
```

1 point

0,5 point

```
        double getY();
        void setX(double newX);
        void setY(double newY);
};



/**********************
Polygone.h
**********************/
#pragma once

#include <vector>
#include "Figure.h"
#include "Point2D.h"

using namespace std;

class Point2D;
class Figure;

#define abs(x) ( (x) >=0 ? (x) : -(x) )

class Polygone : public Figure
{
protected:
        vector<Point2D *> lesSommets;
        bool estFerme;

public:
        Polygone(void);
        static double distance(Point2D &p1,Point2D &p2);
        void insereUnNouveauSommet(Point2D *leSommet, int position = -1);
        void fermeLePolygone();
        double getPerimetre();
        double getSurface();

};



/**********************
Cercle.cpp
**********************/
#include "Cercle.h"


Cercle::Cercle(Point2D centre, double rayon)
{
        this->centre = centre;
        this->rayon = rayon;
}

double Cercle::getPerimetre()
{
        return 2 * M_PI * rayon;
}

double Cercle::getSurface()
{
        return M_PI * rayon * rayon;
}


/**********************
Commande.cpp
**********************/
#include "Commande.h"


Commande::Commande(string identifiantCommande , double lePrixMetreDecoupe , double lePrixMetreCarreMatiere)
{
        this->idCommande = identifiantCommande;
        this->prixMetreDecoupe = lePrixMetreDecoupe;
        this->prixMetreCarreMatiere = lePrixMetreCarreMatiere;
}

void Commande::ajouterNouvelleFigure(Figure* laFigure)
{
        lesFigures.push_back(laFigure);
}

void Commande::cloturerCommande()
{
        commandeTerminee = true;
}
```

1 point

1 point

2,5 points

```
double Commande::getPrix()
{
        double prixP = 0, prixS = 0;
        for (unsigned i = 0; i < lesFigures.size(); i++)
        {
                prixP = lesFigures[i]->getPerimetre() * prixMetreDecoupe;
                prixS = lesFigures[i]->getSurface() * prixMetreCarreMatiere;
        }

        return prixP + prixS;

}



/**********************
main.cpp
**********************/
#include <iostream>
#include <conio.h>
#include "Polygone.h"
#include "Cercle.h"
#include "Commande.h"

using namespace std ;                           // espace de nommage standard


int main()
{
        // Testez la classe Cercle
        Cercle cercle_test({ 0, 0 }, 4);
        cout << "Perimetre du cercle : " << cercle_test.getPerimetre() << "\t Surface du cercle : " << cercle_test.getSurface()
<< endl;



        // Testez la classe Polygone avec la figure de test du sujet
        double Coordonnees[6][2]={ { 1 , 1 } , { 3 , 5 } , { 5 , 7 } , { 5 , 1 } , { 3 , 3 } , { 3 , 1 } };

        Polygone P1;

        for (unsigned i = 0; i < 6 ; i++)
        {
                        P1.insereUnNouveauSommet(new Point2D(Coordonnees[i][0], Coordonnees[i][1]));
        }
        P1.fermeLePolygone();

        cout << "Perimetre : " << P1.getPerimetre() << "\t Surface : " << P1.getSurface();


        cout << endl;


        // Sapin de Noel et boules
        double CoordonneesSapin[15][2]={ { 2 , 2 } , { 5 , 4 } , { 3 , 4} , { 5 , 6 } , { 4 , 6 } , { 6 , 8 },
{ 8 , 6 } , { 7 , 6 },
                                                                { 9 , 4 } , { 7 , 4} , { 10 , 2 } , { 6
.5 , 2 } , { 6.5 , 1 },  { 5.5 , 1 } , { 5.5 , 2 }};


        double CoordonneesCentreCercles[6][2]={ { 2.5 , 3.5 } , { 3.5  , 5.5 } , { 4.5 , 7.5 } , { 7.5  , 7.5 }
, { 8.5 , 5.5 } , { 9.5  , 3.5 } };
        int i;


        // Création du polygone sapin
        Figure *S1;
        S1 = new Polygone;

        for (unsigned i = 0; i < 15; i++)
        {
                ((Polygone *)S1)->insereUnNouveauSommet(new Point2D(CoordonneesSapin[i][0], CoordonneesSapin[i][
1]));
        }
        ((Polygone *)S1)->fermeLePolygone();

        cout <<"superficie du sapin = " << S1->getSurface() << "  " ;
        cout <<"Perimetre du sapin = " << S1->getPerimetre() << endl;

        // Création des 6 cercles


        for (unsigned i = 0; i < 6; i++)
```

Retourner 0 si la commande n'est pas terminée

Test de la classe Cercle : 0,5 point

pg principal de test du polygone de la figure : 2 points

pg principal de test du sapin de Noël : 1,5 points

```
        {
                cout << endl;
                Cercle Boule(Point2D(CoordonneesCentreCercles[i][0], CoordonneesCentreCercles[i][1]), 0.5);
                cout << "superficie du cercle " << i << " = " << Boule.getSurface() << "   ";
                cout << "Perimetre du cercle " << i << " = " << Boule.getPerimetre() << endl;
        }
```

L'objet Boule est perdu est réinitialisé à chaque boucle !

```
        // Création de la commande du Père Noel
        Commande pNoel("Commande du pere noel", 0.26, 12.35);

        // Ajout des figures (le sapin et les 6 cercles) à la commande
        pNoel.ajouterNouvelleFigure(S1 + );

        // Affichage du prix de cette commande
        cout <<"\nCout de la commande : " << ... <<" = " << ... <<" euros" << endl;



        _getch();       // on attend l'appui sur une touche
        return 0 ;      // fin du programme
}



/***********************
Point2D.cpp
***********************/
// Cette classe n'est pas à modifier
#include "Point2D.h"


Point2D::Point2D(double x , double y)
{
        this->x = x;
        this->y = y;
}

double Point2D:: getX()
{ return x ;}

double Point2D::getY()
{
        return y;
}

void Point2D::setX(double newX)
{
        x = newX;
}

void Point2D::setY(double newY)
{
        y = newY;
}


/***********************
Polygone.cpp
***********************/
#include <math.h>
#include "Polygone.h"


Polygone::Polygone(void)
{
        estFerme = false;
}

double Polygone::distance(Point2D &p1, Point2D &p2)
{
        double dist = 0;
        dist = sqrt((p1.getX() - p2.getX()) * (p1.getX() - p2.getX()) + (p1.getY() - p2.getY())*(p1.getY() - p2.
getY()));
        return dist;
}

void Polygone::insereUnNouveauSommet(Point2D *leSommet, int position)
{
        if (position == -1)
        {
```

```
                lesSommets.push_back(leSommet);
        }
        else
        {
                lesSommets.insert(lesSommets.begin() + position, leSommet);
        }
}
```

2 points

```
void Polygone::fermeLePolygone()
{
        estFerme = true;
        lesSommets.push_back(lesSommets[0]);
}

double Polygone::getPerimetre()
{
        double peri = 0;

        if (estFerme == true)
        {
                for (unsigned i = 0; i < lesSommets.size() - 1 ; i++)
                {
                        peri += distance(*lesSommets[i], *lesSommets[i + 1]);
                }
        }
        return peri;
}

double Polygone::getSurface()
{
        double surf = 0;
        if (estFerme == true)
        {
                for (unsigned i = 0; i < lesSommets.size() - 1; i++)
                {
                        surf += (lesSommets[i]->getX() * lesSommets[i + 1]->getY() - lesSommets[i + 1]->getX() *
 lesSommets[i]->getY());
                }
        }
        return abs(surf) * 0.5;
}
```