

Filière d'ingénieur :

Ingénierie Informatique, Big Data et Cloud Computing (II-BDCC)

Département Mathématiques et Informatique

Compte rendu TP3

DE LA CONCEPTION ET LA PROGRAMMATION

ORIENTE OBJET EN C++

Par: BOUZINE Ahmed houssam (II-BDCC1)

Encadré par : M. K. MANSOURI

Année Universitaire : 2022/2023

Exercice 1 :

- 1- Créer une classe *compteur* permettant à tout moment de connaître le nombre d'objets existants.

```
using namespace std;
class Compteur {
public:
    static int counter;
    Compteur();
    ~Compteur();
};
int Compteur::counter = 0;

Compteur::Compteur() {
    ....
    ++counter;
}

Compteur::~~Compteur() {
    ....
    --counter;
}
```

- 2- Ecrire une fonction essai dans laquelle, vous créez deux objets u et v de classe compteur.

```
void essai(){
    Compteur u;
    Compteur v;
}
```

- 3- Ecrire la fonction main dans laquelle, vous créez un objet a, vous appelez ensuite la fonction essai, puis vous créez un objet b.

```
int main() {
    Compteur a;
    essai();
    Compteur b;
    cout << "Le nombre d'objets est n = " << Compteur::counter << endl;
    return 0;
}
```

4- Compiler et exécuter le programme. Conclure.

```
Le nombre d'objets est n = 2
```

Après compilation et exécution on remarque l'affichage du résultat qui 2 car les deux objets qui existent sont seulement a et b les deux autres (u et v) sont détruits.

Exercice 2 :

1- Reprendre la classe *point*, en utilisant la surdéfinition des fonctions membres pour créer plusieurs constructeur de cette classe.

- Premier constructeur : **point()** ; sans paramètres, il initialise x et y à 0,
- Premier constructeur : **point(int)** ; avec un seul paramètre, il initialise x et y à la valeur de ce paramètre,
- Premier constructeur : **point(int, int)** ; avec deux paramètres, il initialise x à la valeur du premier paramètre et y à la valeur du second paramètre,

```
using namespace std;
class Point {
private:
    int x,y;
public:
    Point();
    Point(int);
    Point(int , int);
    void affiche()const;
    void affiche(const char* chaine);
};

Point::Point(){
}
Point::Point(int _x){
    x=_x;
    y=_x;
}
Point::Point(int _x, int _y){
    x=_x;
    y=_y;
}
```

2- Ecrire les deux fonctions membres surchargées suivante :

- Une fonction : **affiche()** ; sans paramètres, permettant d'afficher un point à l'écran,
- Une fonction : **affiche(char *)** ; à un sel paramètre, une chaîne de caractère, permettant le nom du point et d'appeler la fonction : **affiche()** ci dessus.

```
void Point::affiche()const{
    cout << "Les coordonnees du point sont: " << "(" << x << ", " << y << ")" << endl;
}

void Point::affiche(const char* chaine)const{
    cout << chaine << " : ";
    affiche();
}
```

3- Proposer une fonction main() créant des objets de classe point et faisant appel à ces deux fonctions membres.

```
int main() {
    Point point1;
    Point point2(4);
    Point point3(10, 4);

    point1.affiche();
    point2.affiche("p2");
    point3.affiche("p3");

    return 0;
}
```

```
Les coordonnees du point sont: (0,0)
p2 : Les coordonnees du point sont: (4,4)
p3 : Les coordonnees du point sont: (10,4)
```

```
-----
Process exited after 0.0588 seconds with return value 0
Press any key to continue
```

Exercice 3 :

- 1- Reprendre la classe *point*, écrite dans l'exercice 2 ci-dessus en écrivant les différents constructeurs et fonctions membres sous forme de fonctions « *inline* ».

```
using namespace std;
class Point {



private:
    int x,y;
public:
    Point(){
    }
    Point(int _x){
        x=_x;
        y=_x;
    }
    Point(int _x, int _y){
        x=_x;
        y=_y;
    }
    ~Point(){
    }

    void affiche()const{
        cout << "Les coordonnees du point sont: " << "(" << x << "," << y << ")" << endl;
    }

    void affiche(const char* chaine)const{
        cout << chaine << " : ";
        affiche();
    }

};
```

- 2- Comparer la taille du fichier objet « .obj » de cet exemple et celui de l'exemple précédent. Conclure ?

 tp3ex2.o	3/11/2023 10:36 PM	O File	4 KB
 tp3ex3.o	3/11/2023 10:37 PM	O File	6 KB

On remarque que le fichier contenant les fonctions en ligne est plus couteux en terme de mémoire .

Exercice 4 :

Reprenons la classe *point* dans laquelle vous allez introduire une fonction membre nommée « *coïncidence* ». Cette fonction permet de détecter la coïncidence éventuelle entre deux points et qui a comme paramètre un objet de classe *point*.

```
int coincidence(Point point) const {
    return (x == point.x && y == point.y);
}

;

int main() {
    Point point1;
    Point point2(4);
    Point point3(10, 4);

    point1.affiche();
    point2.affiche("p2");
    point3.affiche("p3");
    if (point1.coincidence(point2)) {
        std::cout << "Les points p1 et p2 sont coïncidents." << std::endl;
    }
}
```

Exercice 5 : Passage de paramètres par adresse :

- 1- Modifier la fonction membre « *coïncidence* » de l'exemple précédent de sorte que son prototype devienne *int point::coïncidence (point *adpt)*,

```
int coincidence(Point *point) const {
    return (x == point->x && y == point->y);
}
```

- 2- Ré-écrire le programme principal en conséquence.

```
int main() {
    Point point1;
    Point point2(4);
    Point point3(10, 4);

    point1.affiche();
    point2.affiche("p2");
    point3.affiche("p3");
    if (point1.coincidence(&point2)) {
        std::cout << "Les points p1 et p2 sont coïncidents." << std::endl;
    } else {
        std::cout << "Les points p1 et p2 ne sont pas coïncidents." << std::endl;
    }
    return 0;
}
```

Exercice 6 : Passage de paramètres par référence :

- 1- Modifier la fonction membre « *coïncidence* » de l'exemple précédent de sorte que son prototype devienne *int point::coïncidence (point &adpt)*,

```
int coincidence(Point& point) const {  
    return (x == point.x && y == point.y);  
}
```

- 2- Ré-écrire le programme principal en conséquence.

```
int main() {  
    Point point1;  
    Point point2(4);  
    Point point3(10, 4);  
  
    point1.affiche();  
    point2.affiche("p2");  
    point3.affiche("p3");  
    if (point1.coïncidence(point2)) {  
        std::cout << "Les points p1 et p2 sont coïncidents." << std::endl;  
    } else {  
        std::cout << "Les points p1 et p2 ne sont pas coïncidents." << std::endl;  
    }  
    return 0;  
}
```

Exercice 7 :

- 1- Mettre cette classe en oeuvre dans un programme principal *void main()*, en ajoutant une fonction membre *float det(vecteur)* qui retourne le déterminant des deux vecteurs (celui passé en paramètre et celui de l'objet),

```
float vecteur::det(vecteur vt)  
{  
    return x * vt.y - y * vt.x;  
}
```

```
int main()  
{  
    vecteur u(5, 3);  
    vecteur v(9, 7);  
  
    cout << "Determinant : " << u.det(v) << "\n";  
  
}
```

2- Modifier la fonction *déterminant* de sorte de passer le paramètre par adresse.

```
float vecteur::det_adresse(vecteur *vt)
{
    return x * vt->y - y * vt->x;
}
```

Appel :

```
cout << "Determinant (fonction adresse) : " << u.det_adresse(&v) << "\n";
```

3- Modifier la fonction déterminant de sorte de passer le paramètre par référence.

```
float vecteur::det_reference(vecteur &vt)
{
    return x * vt.y - y * vt.x;
}
```

Appel :

```
cout << "Determinant (fonction reference) : " << u.det_reference(v) << "\n";
```

Exercice 8 : Classe vecteur 2 :

1- Modifier la fonction « homothétie » qui retourne le vecteur modifié par valeur. (prototype: vecteur vecteur::homotethie(float val)).

```
vecteur vecteur::homotethie(float val)
{
    x = x * val;
    y = y * val;
    return {x,y};
}
```

2- Modifier la fonction « homothétie » qui retourne le vecteur modifié par adresse.

```
vecteur* vecteur::homotethie_adresse(float val)
{
    x = x * val;
    y = y * val;
    return this;
}
```


- 3- Modifier la fonction « homotéthie » qui retourne le vecteur modifié par référence.

```
vecteur& vecteur::homotethie_reference(float val)
{
    x = x * val;
    y = y * val;
    return *this;
}
```