

ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE
DE MOHAMMEDIA

➤ UNIVERSITÉ HASSAN II DE CASABLANCA

TP3 : programmation orientée objet en c++

ETUDIANT DE FILIER GLSID 1
ENSET

➤ ENCADRÉ PAR:
M.K.MANSOURI

➤ REALISER PAR :
ZAKARIA EL MOURTAZAK

EXERCICE 1

LE CODE

```
#include <iostream>

class Compteur
{
private:
    static int ctr;
public:
    Compteur();
    ~Compteur();
};

int Compteur::ctr = 0;
Compteur::Compteur()
{
    std::cout << "Un objet vient de se détruire : "
                << "\n";
    std::cout << "Il reste maintenant : " << ++ctr << " Objets";
    getchar();
}

Compteur::~~Compteur()
{
    std::cout << "Un objet vient de se détruire : "
                << "\n";
    std::cout << "Il reste maintenant : " << --ctr << " Objets";
    getchar();
}

void essai()
{
    Compteur u, v;
}

int main(int argc, char const *argv[])
{
    Compteur a;
    essai();
    Compteur b;
    return 0;
}
```

L'EXECUTION

```
, if ($?) { .\exercice1 }
Un objet vient de se d|@truire :
Il reste maintenant : 1 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 2 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 3 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 2 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 1 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 2 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 1 Objets
Un objet vient de se d|@truire :
Il reste maintenant : 0 Objets
```

On déduit que les variables statiques dans une classe : comme les variables déclarées comme statiques ne sont initialisées qu'une seule fois car elles se voient allouer de l'espace dans un stockage statique séparé, les variables statiques d'une classe sont partagées par les objets. Il ne peut pas y avoir plusieurs copies des mêmes variables statiques pour différents objets. C'est également pour cette raison que les variables statiques ne peuvent pas être initialisées à l'aide de constructeurs.

EXERCICE 2

LE CODE

```
void Point::afficher()
{
    gotoxy(x, y);
    std::cout << "*";
}

void Point::afficher(const char *nom)
{
    gotoxy(x, y);
    afficher();
    std::cout << nom;
    afficher();
}

void gotoxy(short a, short b)
{
    COORD pos = {a, b};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

main(int argc, char const *argv[])
{
    Point p1, p2(1), p3(1, 2);
    system("cls");
    p1.afficher();
    getchar();

    p2.afficher("p2");
    getchar();

    p3.afficher("p3");
    getchar();
}
```

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

class Point
{
private:
    int x, y;

public:
    Point():x(0),y(0){};
    Point(int x):x(x),y(0){};
    Point(int x, int y):x(x),y(y){};
    void afficher();
    void afficher(const char *);
};

void gotoxy(short a, short b);
```

L'EXECUTION

**p2

*p3

EXERCICE 3





1.

LE CODE

```
class Point
{
private:
    int x, y;

public:
    Point()
    {
        x = 0;
        y = 0;
    };
    Point(int)
    {
        this->x = x;
        y = 0;
    };
    Point(int, int)
    {
        this->x = x;
        this->y = y;
    };
    void afficher();
    void afficher(const char *);
};
```

2.

 exercise2.cpp	4/5/2022 3:27 AM	C++ Source File	1 KB
 exercise2.o	4/5/2022 3:28 AM	O File	46 KB
 exercise3.cpp	4/5/2022 3:16 AM	C++ Source File	1 KB
 exercise3.o	4/5/2022 3:28 AM	O File	44 KB

On déduit que les fonctions **inline** indiquent au compilateur que chaque appel à la fonction **inline** devra être remplacé par le corps de cette fonction. Afin de générer un exécutable de taille raisonnable

EXERCICE 4

LE CODE

```
class Point
{
private:
    int x, y;

public:
    Point();
    Point(int);
    Point(int, int);
    void afficher();
    void afficher(const char *);
    int coincidence(Point);
};
```

```
int Point::coincidence(Point p)
{
    return ((x == p.x) && (y == p.y));
}
```

EXERCICE 5

1.

LE CODE

```
int Point::coincidence(Point *p)
{
    return this == p;
}
```

2.

```

1  main(int argc, char const *argv[])
2  {
3      Point p1, p2(1, 2);
4      (p1.coïncidence(&p2) == 1) ? (std::cout << "les deux points p1 et p2 coïncident \n") :
5      (std::cout << "les deux points p1 et p2 ne coïncident pas\n");
6      getchar();
7      (p1.coïncidence(&p1) == 1) ? (std::cout << "les deux points p1 et p1 coïncident\n") :
8      (std::cout << "les deux points p1 et p2 ne coïncident pas\n");
9      getchar();
10 }

```

L'EXECUTION

```
les deux points p1 et p2 ne coïncident pas
les deux points p1 et p1 coïncident

```

EXERCICE 6

1.

LE CODE

```
int Point::coincidence(Point &p)
{
    return this == &p;
}
```

2.

LE CODE

```
main(int argc, char const *argv[])
{
    Point p1, p2(0);
    (p1.coincidence(p2) == 1) ? (std::cout << "les deux points p1 et p2 coïncident") :
    (std::cout << "les deux points p1 et p2 ne coïncident pas");
    getchar();
    (p1.coincidence(p1) == 1) ? (std::cout << "les deux points p1 et p1 coïncident\n") :
    (std::cout << "les deux points p1 et p2 ne coïncident pas\n");
    getchar();
}
```

L'EXECUTION

```
les deux points p1 et p2 ne coïncident pas  
les deux points p1 et p1 coïncident  
□
```

EXERCICE 7

1.

LE CODE

```
1  #include <iostream>
2
3  class Vecteur
4  {
5  private:
6      float x, y;
7
8  public:
9      Vecteur(float, float);
10     void homotethie(float);
11     void afficher();
12     float det(Vecteur);
13     float getX()
14     {
15         return x;
16     };
17     float getY()
18     {
19         return y;
20     };
21 };
22
23 Vecteur::Vecteur(float abs = 0, float ord = 0)
24 {
25     x = abs;
26     y = ord;
27 }
28
29 void Vecteur::homotethie(float val)
30 {
31     x *= val;
32     y *= val;
33 }
34
35 float Vecteur::det(Vecteur v1)
36 {
37     return ((x * v1.y) - (y * v1.x));
38 }
39
40 int main(int argc, char const *argv[])
41 {
42     Vecteur v1(1, 2);
43     Vecteur v2(3, 4);
44     std::cout << "le determinant des deux vecteur v1: x = " << v1.getX() << ", y = " <<
45     v1.getY() << " et v2: x = " << v2.getX() << ", y = " << v2.getY() << " est " << v1.det(v2) << std::endl;
46     return 0;
47 }
```


2.

LE CODE

```
float Vecteur::det(Vecteur *v1)
{
    return ((x * v1->y) - (y * v1->x));
}

int main(int argc, char const *argv[])
{
    Vecteur v1(1, 2);
    Vecteur v2(3, 4);
    std::cout << "le determinant des deux vecteur v1: x = " << v1.getX() << ", y = " << v1.getY()
    << " et v2: x = " << v2.getX() << ", y = " << v2.getY() << " est " << v1.det(&v2) << std::endl;
    return 0;
}
```

3.

LE CODE

```
float Vecteur::det(Vecteur &v1)
{
    return ((x * v1.y) - (y * v1.x));
}

int main(int argc, char const *argv[])
{
    Vecteur v1(1, 2);
    Vecteur v2(3, 4);
    std::cout << "le determinant des deux vecteur v1: x = " << v1.getX() << ", y = " << v1.getY()
    << " et v2: x = " << v2.getX() << ", y = " << v2.getY() << " est " << v1.det(v2) << std::endl;
    return 0;
}
```

L'EXECUTION

```
le determinant des deux vecteur v1: x = 1, y = 2 et v2: x = 3, y = 4 est -2
PS C:\Users\user\Documents\TP_CPP_ELMOURTAZAK_ZAKARIA\TPS\Chapitre 3> █
```

EXERCICE 8

1.

LE CODE

```
Vecteur Vecteur::homotethie(float val)
{
    Vecteur res;
    res.x = x * val;
    res.y = y * val;
    return res;
}

int main(int argc, char const *argv[])
{
    Vecteur v1(1, 2);
    Vecteur v2 = v1.homotethie(2);
    v2.afficher();
    return 0;
}
```

2.

LE CODE

```
Vecteur *Vecteur::homotethie(float val)
{
    Vecteur *res = new Vecteur;
    res->x = x * val;
    res->y = y * val;
    return res;
}

int main(int argc, char const *argv[])
{
    Vecteur v1(1, 2);
    Vecteur v2 = *v1.homotethie(2);
    v2.afficher();
    return 0;
}
```

3.

LE CODE

```
Vecteur &Vecteur::homotethie(float val)
{
    static Vecteur res;
    res.x = x * val;
    res.y = y * val;
    return res;
}

int main(int argc, char const *argv[])
{
    Vecteur v1(1, 2);
    Vecteur v2 = v1.homotethie(2);
    v2.afficher();
    return 0;
}
```

L'EXECUTION

```
x = 2, y = 4
```