C++ PoP MT & EL /White vendredi 11 avril 2025

Ne PAS retourner ces feuilles avant d'en être autorisé!

Merci de poser votre carte CAMIPRO en évidence sur la table.

Vous pouvez déjà compléter et lire les informations ci-dessous:

NOM en MAJUSCULE _____

Prénom en MAJUSCULE _____

Numéro SCIPER _____

Signature _____

BROUILLON : Ecrivez aussi votre NOM-Prénom sur la feuille de brouillon fournie. Toutes vos réponses doivent être sur cette copie d'examen. Les feuilles de brouillon sont ramassées pour être immédiatement détruites.

Le test écrit commence à: **14h15**

Les copies d'examens sont ramassées à : **15h45**

Le contrôle de C++ PoP reste SANS appareil électronique

Vous avez le droit d'avoir tous vos documents **personnels** sous forme papier: dictionnaire, livres, cours, exercices, code, projet, notes manuscrites, etc...

Vous pouvez utiliser un crayon à papier et une gomme

Ce contrôle écrit de C++ PoP permet d'obtenir **35 points** sur un total de 100 points pour le cours complet.

1) (9 pts) Divers usages de `static` :

Le code **ex1.cc** compile sans warning avec -std=c++11 -Wall. Son exécution se termine normalement.

```
1   #include <iostream>
2   using namespace std;
3
4   static int compteur = 100;
5
6   class X
7   {
8         static int compteur;
9         int id;
10  public:
11        X() : id(compteur)
12        {
13              compteur++;
14              cout << "Constructeur X (id = " << id
15                    << "), compteur = "
16                    << compteur << endl;
17        }
18        ~X()
19        {
20              cout << "Destructeur X (id = " << id
21                    << "), compteur = "
22                   << compteur << endl;
23              compteur--;
24        }
25        void afficher() const
26        {
27              cout << "Objet X d'id = " << id
28                    << " | valeur actuelle de compteur = "
29                    << compteur << endl;
30        }
31  };
32
33  int X::compteur(200);
34
35  int main()
36  {
37        compteur = 33;
38        {
39              X x1;
40              X x2;
41              x1.afficher();
42        }
43        X x3;
44        x3.afficher();
45        return 0;
    }
```

L'exécution de ce code produit plusieurs affichages dans le terminal. Utiliser la page suivante en indiquant dans la colonne de gauche le numéro de la ligne de code causant l'affichage et la colonne de droite pour montrer et JUSTIFIER le texte affiché.

N° ligne de code	Texte affiché et JUSTIFICATION

2) (9 pts) Héritage : soit le fichier **ex2.cc** dont le code est listé ci-dessous
Ce code **ex2.cc** compile sans warning avec -std=c++11 -Wall. Son exécution se termine normalement.

```
1   #include <iostream>
2   using namespace std;
3
4   class Sensor {
5   public:
6         Sensor( int id ) : id( id ) {
7               cout << "init Sensor : " << id << endl;
8         }
9         virtual ~Sensor() {
10              cout << "destroy Sensor : " << id << endl;
11        }
12        virtual void report() const {
13              cout << "Sensor : " << id << " reporting " << endl;
14        }
15  protected:
16        int id;
17  };
18
19  class TemperatureSensor : public Sensor {
20  public:
21        TemperatureSensor( int id, float temp )
22              : Sensor( id ), temperature( temp ) {
23              cout << "init temperature : "
24                    << temperature << "°C" << endl;
25        }
26        ~TemperatureSensor() {
27              cout << "destroy temperature sensor " << id << endl;
28        }
29        void report() const override {
30              cout << "temperature of " << id << " is "
31                    << temperature << "°C" << endl;
32        }
33  private:
34        float temperature;
35  };
36
37  void sendReport( const Sensor& s ) {
38        s.report();
39  }
40
41  int main() {
42
43        Sensor* s = new TemperatureSensor( 2025, 11.2 );
44
45        s->report();
46
47        sendReport( *s );
48
49        delete s;
50
51        return 0;
52  }
53
```

2.1) JUSTIFIER tous les affichages obtenus par son exécution **:**

2.2) On ajoute un constructeur supplémentaire entre les lignes **25** et **26** avec le code suivant :

```
TemperatureSensor( int id, float temp )
     : id( id ), temperature( temp ) {
     cout << "init temperature : "
          << temperature << "°C" << endl;
}
```

Question : Indiquer si la compilation de ce programme produit toujours un exécutable :
Si vous répondez « oui », préciser s'il y modification de l'affichage obtenu à l'exécution.
Si vous répondez « non », préciser la cause de l'erreur

2.3) Revenons au code initial de la page précédente. On enlève le mot **`virtual`** de la ligne 9.
Question : Indiquer si la compilation de ce programme produit toujours un exécutable :
Si vous répondez « oui », préciser s'il y modification de l'affichage obtenu à l'exécution.
Si vous répondez « non », préciser la cause de l'erreur

3) (9 pts) Héritage Multiple

Ce code **ex3.cc** compile sans warning avec -std=c++11 -Wall. Son exécution se termine normalement.

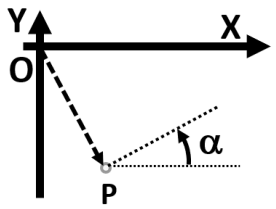
```
1   #include <iostream>
2   using namespace std;
3   class Base {
4   public :
5         Base() {cout << " Base constructor " <<  endl ;}
6         virtual ~ Base() {cout << " Base destructor " <<  endl ;}
7         virtual void display(){cout << " Base display " <<  endl ;}
8         void show(){cout << " Base show " << endl ;}
9   };
10
11  class Derived1 : public virtual Base {
12  public :
13        Derived1() {cout << " Derived1 constructor " <<  endl ;}
14        ~ Derived1() {cout << " Derived1 destructor " <<  endl ;}
15        void display()override {cout << " Derived1 display "
16                                     <<  endl ;}
17        void show(){cout << " Derived1 show " <<  endl ;}
18  };
19
20  class Derived2 : public virtual Base {
21  public :
22        Derived2() {cout << " Derived2 constructor " <<  endl ;}
23        ~ Derived2() {cout << " Derived2 destructor " <<  endl ;}
24        void display()override {cout << " Derived2 display "
25                                     <<  endl ;}
26        void show(){cout << " Derived2 show " <<  endl ;}
27  };
28
29  class Final : public Derived1 , public Derived2 {
30  public :
31        Final() {cout << " Final constructor " <<  endl ;}
32        ~ Final() {cout << " Final destructor " <<  endl ;}
33        void display()override {cout << " Final display " <<  endl ;}
34  };
35
36  int main()
37  {
38      {
39          cout << "Block 1:" <<  endl ;
40          Base* b = new Derived1();
41          b->display();
42          b->show();
43          delete b;
44      }
45
46      {
47          cout << "Block 2:" <<  endl ;
48          Final f;
49          Base& ref = f;
50          ref.display();
51          ref.show();
52       }
53       return 0;
54  }
```

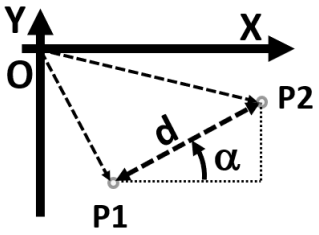
L’exécution de ce code produit plusieurs affichages dans le terminal. Utiliser cette table en indiquant dans la colonne de gauche le numéro de la ligne de code causant l’affichage et la colonne de droite pour montrer et JUSTIFIER le texte affiché.

N° ligne de code	Texte affiché et *JUSTIFICATION*
39	**`Block 1 :`** *Simple affichage d’une chaîne constante*
47	**`Block 2 :`** *Simple affichage d’une chaîne constante*

4) (8 pts) surcharge d'opérateur : soit le fichier **ex4.cc** dont le code partiel est listé ci-dessous. On y voit la définition d'une structure **S2d** pour placer un point (x,y) dans le plan 2D et d'une classe **MovingPoint** pour modéliser un point **p** du plan (ligne 16) qui se déplace selon la direction indiquée par l'angle **alpha** en rd (ligne 17 et illustration sur la droite) :

```
1   #include <cmath>
2
3   struct S2d{
4       S2d(double x=0., double y=0.): x(x),y(y){}
5       double x ;
6       double y ;
7   };
8
9   class MovingPoint {
10  public:
11        MovingPoint() = default;
12        MovingPoint(S2d p, double alpha=0)
13                    :p(p),alpha(alpha) {}
14        MovingPoint operator+(double d);
15  private:
16        S2d p;
17        double alpha;
18  };
19
20  // il manque la définition de la surcharge de l'opérateur +
21
22  int main(){
23        S2d p(2., -4);
24        MovingPoint p1(p,0.3);
25        MovingPoint p2;
26        p2 = p1 + 5. ;
27        return 0;
28  }
```





On demande d'écrire le code source de la surcharge de l'opérateur de l'addition dont le prototype est visible ligne 14. A titre d'exemple, cette surcharge est utilisée ligne 26 pour mettre à jour la valeur de l'instance **p2** obtenue en déplaçant **p1** dans la direction indiquée par **alpha** d'une distance **d** comme illustré avec la seconde figure.

```
MovingPoint MovingPoint::operator+(double d)
{









}
```