C++ PoP MT & EL /White vendredi 5 mai 2023

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: **9h15**

Les copies d'examens sont ramassées à : **10h45**

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) (7 pts) Classe : soit le fichier **ex1.cc** dont le code est listé ci-dessous

```
1   #include <iostream>
2
3   using namespace std;
4
5   class A
6   {
7       static int a;
8   public:
9       void show()
10      {
11          a++;
12          cout<<"a: "<<a<<endl;
13      }
14  };
15
16  int A::a = 5;
17
18  int main(int argc, char* argv[])
19  {
20       A a;
21       return 0;
22  }
```

1.1) On compile le fichier ex1.cc avec la commande : **`g++ -std=c++11  ex1.cc -o ex1`**

Choisir une réponse parmi les suivantes

Réponse	Description	**Cocher une seule case**
A	L'exécution affiche : **`a: 5`**	
B	L'exécution affiche : **`a: 6`**	
C	L'exécution se termine normalement sans rien afficher	
D	L'exécution se termine anormalement, par exemple avec un message **`segmentation fault`**	
E	Une erreur est détectée à l'étape de la **compilation**	

1.2) **Justifier votre réponse** à la question précédente ; si vous avez choisi les réponses D ou E alors précisez la nature de l'erreur et les instructions qu'il faut modifier ou ajouter pour supprimer le problème.

2) (7 pts) Correction de code :

Ce code produit une erreur à la compilation en C++11.

```
1   #include <iostream>
2   using namespace std;
3
4   class A {
5   public:
6         A();
7         A( int );
8         int getVal() { return val; }
9   protected:
10        int val;
11  };
12
13  A::A() {
14        cout << "Call A::A() constructor." << endl;
15        val = 0;
16  }
17
18  A::A( int i ) {
19        cout << "Call A::A( int ) constructor." << endl;
20        val = i;
21  }
22
23  class B {
24        A a;
25  public:
26       B():a(1){}
27        void show() {cout << val << endl ; }
28  };
29
30  int main() {
31      B b;
32      b.show();
33      return 0;
34  }
```

2.1) Quelle est la nature de l'erreur produisant l'erreur à la compilation ?

2.2) Corriger le code de manière à pouvoir exécuter la fonction main() (**on ne peut pas modifier la fonction main())** . Indiquer ce que vous ajoutez/supprimez ; préciser où vous faites les modifications.

2.3) Montrer ce qui est affiché avec l'exécution de main avec le code corrigé.

3) (7 pts) surcharge des opérateurs pour faciliter les opérations de cryptographie

L'arithmétique modulaire sur des entiers non-signés est un élément central pour certaines opérations de cryptographie. L'addition et la multiplication sont effectuées comme avec les entiers mais, en plus, on applique l'opérateur modulo (%) au résultat.

Exemple : l'addition **5+7** donne **12** en arithmétique standard tandis qu'en arithmétique modulaire modulo **11** on obtient **1** comme résultat car **12%11** vaut **1**.

Cet exercice propose une classe **`Zn`** pour mémoriser une valeur **`val`** dans l'arithmétique modulaire modulo **`mod`**. Le code ci-dessous surcharge les opérateurs **+** et ***** pour effectuer ces opérations en arithmétique modulaire modulo **`mod`**.

Ce code compile sans warning avec les options -std=c++11 -Wall

```
1   #include <iostream>
2   using namespace std;
3
4   class Zn
5   {
6   public:
7       Zn(unsigned val, unsigned mod):val{val%mod},mod{mod}{}
8   private:
9       unsigned int val;
10      unsigned int mod;
11      friend ostream& operator<<(ostream& os, const Zn& zn);
12      friend Zn operator+(const Zn& lhs,const Zn& rhs);
13      friend Zn operator*(const Zn& lhs,const Zn& rhs);
14  };
15
16  Zn operator+(const Zn& lhs,const Zn& rhs)
17  {
18      return Zn((lhs.val+rhs.val) % lhs.mod,lhs.mod);
19  }
20  Zn operator*(const Zn& lhs,const Zn& rhs)
21  {
22      return Zn((lhs.val*rhs.val) % lhs.mod,lhs.mod);
23  }
24  ostream& operator<<(ostream& os, const Zn& zn)
25  {
26      os << "Zn("<<zn.val<<" mod "<<zn.mod<<")";
27      return os;
28  }
29
30  int main()
31  {
32        Zn a(3,7);
33        Zn b(6,11);
34        Zn c(5,13);
35        cout << "a="<<a<<" b="<<b <<" c="<<c<< endl; // Q1
36        cout << a+b+c << endl;                       // Q2
37        cout << a*b*c << endl;                       // Q3
38        cout << a+b*c << endl;                       // Q4
39  }
```

Rappel : les règles de priorité entre opérateurs s'appliquent aussi aux opérateurs surchargés.
Chaque question est liée à certaines lignes du code indiquées par un commentaire en fin de ligne.

3.1) Quel est l'affichage réalisé à la ligne avec le commentaire //Q1

3.2) Justifiez comment l'affichage de la question 3.1 est obtenu en détaillant la suite des appels

3.3) Quel est l'affichage réalisé à la ligne avec le commentaire //Q2

3.4) Justifiez comment l'affichage de la question 3.3 est obtenu est détaillant les appels et les valeurs intermédiaires des calculs.

3.5) Quel est l'affichage réalisé à la ligne avec le commentaire //Q3

3.6) Justifiez comment l'affichage de la question 3.5 est obtenu est détaillant les appels et les valeurs intermédiaires des calculs.

3.7) Quel est l'affichage réalisé à la ligne avec le commentaire //Q4

3.8) Justifiez comment l'affichage de la question 3.7 est obtenu est détaillant les appels et les valeurs intermédiaires des calculs.

4) (7 pts) Héritage : soit le fichier **ex4.cc** dont le code est listé ci-dessous

```
1   #include <iostream>
2
3   class Vehicle
4   {
5   public:
6       Vehicle(int wheels) : wheels_(wheels) {}
7
8       int getWheels() const
9       {
10          return wheels_;
11      }
12
13  private:
14      int wheels_;
15  };
16
17  class Car : public Vehicle
18  {
19  public:
20      Car() : Vehicle(4) {}
21
22      void honk()
23      {
24          std::cout << "Beep! Beep!" << std::endl;
25      }
26  };
27
28  int main()
29  {
30      Car myCar;
31      std::cout << "My car has " << myCar.getWheels()
32                                 << " wheels." << std::endl;
33      myCar.honk();
34
35      Vehicle myVehicle;
36      std::cout << "My vehicle has " << myVehicle.getWheels()
37                                     << " wheels." << std::endl;
38
39      return 0;
40  }
```

4.1) On compile le fichier ex4.cc avec la commande : **`g++ -std=c++11  ex4.cc -o ex4`**
Cette commande produit une erreur de compilation.

Quelle ligne de code / quelle instruction produit cette erreur ? Justifier votre réponse

4.2) Corriger la cause de l'erreur de compilation (**sans modifier le code de main()**). Vous pouvez au choix, soit modifier une ligne (*indiquer le numéro de cette ligne et recopiez ci-dessous la version correcte de la ligne*), soit ajouter du code supplémentaire (*indiquer où ce code doit être inséré et écrire ce code ci-dessous*).

4.3) Quel est l'affichage qui doit être produit par l'exécution après correction de l'erreur de compilation ?

5) (7 pts) Héritage multiple : soit le fichier **ex5.cc** dont le code est listé ci-dessous

```
1   #include <iostream>
2
3   class Operable {
4   public:
5       Operable() : id(0) {}
6       virtual std::string getName() const = 0;
7       int getId() const {return id;}
8   protected:
9       int id;
10  };
11
12  class Movable : public Operable {
13  public:
14      virtual void move() const = 0;
15  };
16
17  class Drawable : public Operable {
18  public:
19      virtual void draw() const = 0;
20  };
21
22  class Shape : public Movable, public Drawable {
23  public:
24      Shape(std::string name) : name_(name) {}
25      void move() const override {
26          std::cout << getName() << " is moving."<< std::endl;
27      }
28      void draw() const override {
29          std::cout << getName() << " is being drawn."<< std::endl;
30      }
31      std::string getName() const override {
32          return name_;
33      }
34  protected:
35      std::string name_;
36  };
37
38  int main() {
39      Shape *myShape = new Shape("Circle");
40      Movable *movablePtr = myShape;
41      Drawable *drawablePtr = myShape;
42
43      std::cout << myShape->getName() << std::endl;
44      std::cout << myShape->getId()   << std::endl;
45      movablePtr->move();
46      drawablePtr->draw();
47
48      delete myShape;
49      return 0;
50  }
```

5.1) On compile le fichier ex5.cc avec la commande : **`g++ -std=c++11  ex5.cc -o ex5`**
Cette commande produit une erreur de compilation.

Quelle ligne de code / quelle instruction produit cette erreur ? Justifier votre réponse
(répondre en haut de la page suivante)

5.2) Corriger la cause de l'erreur de compilation (**sans modifier le code de main()**) ; préciser les numéros de ligne et écrire ci-dessous les lignes corrigées entières (*max deux lignes de code*)

5.3) Quel est l'affichage qui doit être produit par l'exécution après correction de l'erreur de compilation ?

5.4) A quoi sert le mot clef **`override`** visible sur les lignes 25, 28 et 31 ?

5.5) Le fichier corrigé compile-t-il si on supprime le mot clef **`override`** sur les lignes 25, 28 et 31 ?

5.6) Avec cette version corrigée du code, on recompile le code mais cette fois on demande l'affichage des warning avec : **`g++ -std=c++11  ex5.cc -Wall -o ex5`**
Un warning est affiché pour la ligne 48. Pour quelle raison ce warning est-il affiché ?

5.7) **Sans modifier le code de main(),** que faut-il ajouter de plus pour faire disparaitre le warning ? indiquer où cet ajout est effectué.