C++ PoP MT & EL /White vendredi 17 mai 2024

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) (6 pts) surcharge des opérateurs : soit le fichier **ex1.cc** dont le code est listé ci-dessous

```
1   #include <iostream>
2   #include <string>
3
4   class Myclass {
5   public:
6     Myclass(int xx, const std::string &ss): x(xx), s(ss) {c++;}
7     static int get_c() {return c;}
8   private:
9     static int c;
10    int x;
11    std::string s;
12  };
13
14  int Myclass::c = 0;
15
16  Myclass operator+(const Myclass &obj1, const Myclass &obj2) {
17
18    return Myclass(obj1.x + obj2.x, obj1.s + obj2.s);
19  }
20
21  std::ostream &operator<<(std::ostream &stream, const Myclass &obj) {
22
23    return stream << "x: " << obj.x << " s: " << obj.s;
24  }
25
26  int main() {
27
28    Myclass ob1(10, "10"), ob2(20, "20"), ob3(30, "30"), ob4(40, "40");
29
30    std::cout << ob1 + ob3 + ob4 + ob2 << std::endl;
31
32    std::cout << Myclass::get_c() << std::endl;
33
34    return 0;
35  }
```

1.1) On compile le fichier ex1.cc avec la commande : **`g++ -std=c++11  ex1.cc -o ex1`**
Cette commande détecte des erreurs de compilation dès la ligne 18.
Quelle est la cause de ces erreur ?

1.2) Il existe plusieurs manières de corriger ces erreurs SANS MODIFIER la fonction main() et tout en respectant le but des surcharges d'opérateurs. On demande de décrire, sur la page suivante, deux approches indépendantes qui suppriment ces erreurs (note : il en existe plus de deux).

Pour cela, indiquez la ou les lignes de code que vous modifiez ou que vous ajoutez. En cas d'ajout, indiquer où vous ajoutez du code « entre les lignes xx et yy » du code visible en page 2.

1.2.1.) Correction 1 :

1.2.2.) Correction 2 :

1.3) Quel est le résultat de l'exécution de ce programme après correction des erreurs.

1.3.1) JUSTIFIER chaque valeur affichée par l'exécution de la **ligne 30**. Rappel : les règles de priorité entre opérateurs s'appliquent aussi aux opérateurs surchargés.

1.3.2) JUSTIFIER la valeur affichée par l'exécution de la **ligne 32**

2) (6 pts) Analyse de code :

Le code **ex2.cc** compile avec succès en C++11 sans produire de warning. Pourtant, après l'affichage de quelques lignes, l'exécution se termine par un message signalant un problème.

```
1   #include <iostream>
2
3   class A {
4   public:
5     A(int new_x): px(new int(new_x)) {}
6     ~A()          {delete px; }
7     int f() const {return 1 + *px; }
8     int* getpx()  {return px; }
9
10  protected:
11    int *px;
12  };
13
14  class B : public A {
15  public:
16    B(): A(5) {}
17    ~B() {}
18    int f() const {return 2 + *px; }
19  };
20
21  int main() {
22    A a1(39);
23    A a2(a1);
24
25    std::cout << *a2.getpx() << std::endl;
26
27    (*a2.getpx()) += 2;
28    std::cout << *a2.getpx() << std::endl;
29
30    std::cout << ++(*a1.getpx()) << std::endl;
31
32    B b;
33    A a3 = b;
34    std::cout << a3.f() << std::endl;
35
36    return 0;
37  }
```

2.1) **ligne 24** : dessiner l'état des variables **a1** et **a2** avec la représentation « boîte et flèche » après leur initialisation :

2.2) **ligne 25** : JUSTIFIER la valeur affichée

2.3) **ligne 28** : JUSTIFIER la valeur affichée

2.4) **ligne 30** : JUSTIFIER la valeur affichée

2.5) **ligne 34** : dessiner l'état des variables **b** et **a3** avec la représentation « boîte et flèche » après leur initialisation et JUSTIFIER la valeur affichée

2.6) proposer une explication concernant l'obtention d'un message d'erreur après l'exécution des affichages précédents

3) (6 pts) une petite dose de vector

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

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   class MyClass {
7   public:
8       MyClass(int id) : id_(id) {
9           cout << "Constructeur pour ID: " << id_ << endl;
10      }
11      ~MyClass() {
12           cout << "Destructeur pour ID: " << id_ << endl;
13      }
14      int getID() const { return id_; }
15  private:
16      int id_;
17  };
18
19  int main() {
20      vector<MyClass *> myVector;
21
22      for (int i = 1; i <= 3; ++i) {
23          myVector.push_back(new MyClass(i));
24      }
25
26      for (MyClass *ptr : myVector) {
27          cout << "ID: " << ptr->getID() << endl;
28      }
29      myVector.clear();
30
31      return 0;
32  }
```

3.1) JUSTIFIER les affichages obtenus par son exécution

3.2) PRECISER l'impact de la **ligne 29** ; comment appelle-t-on le problème qui pourrait en résulter pour un programme plus ambitieux ayant les lignes 22-29 dans une boucle **`while(true)`** ?

4) (5 pts) un petit héritage : soit le fichier **ex4.cc** dont le code est listé ci-dessous

Ce code **ex4.cc** compile sans warning avec -std=c++11 -Wall. , Son exécution se termine normalement.

```
1   #include <iostream>
2   #include <string>
3
4   using namespace std;
5
6   class Pet {
7   public:
8       Pet(const string& name = "animal"){
9           cout << "Created " << name << "." << endl;
10      }
11      virtual ~Pet() {}
12      virtual void makeSound(){
13          cout << "Generic pet sound" << endl;
14      }
15  };
16
17  class Dog : public Pet {
18  public:
19      Dog() : Pet("dog") {}
20      void makeSound() override { cout << "Woof!" << endl; }
21  };
22
23  int main() {
24      Dog myDog;
25      Pet pet(myDog);
26      pet.makeSound();
27      return 0;
28  }
```

4.1) JUSTIFIER tout ce qui est affiché par son exécution

5) (6 pts) un autre héritage : soit le fichier **ex5.cc** dont le code est listé ci-dessous

Ce code **ex5.cc** compile sans warning avec -std=c++11 -Wall. , Son exécution se termine normalement.

```
1   #include <iostream>
2   #include <array>
3   using namespace std;
4
5   class Animal {
6   public:
7       Animal(string name) : name(name) {}
8       virtual ~Animal() {}
9       virtual void speak() const = 0 ;
10      void greet() const { cout << "Hello, I am " << name << endl; }
11      const string& getName() const { return name; }
12  private:
13      string name;
14  };
15
16  class Lion : public Animal {
17  public:
18      Lion() : Animal("Leo") {}
19      void speak() const { cout << "Roar" << endl; }
20  };
21
22  class Tiger : public Animal {
23  public:
24      Tiger() : Animal("Simba") {}
25      void speak() const { cout << "Moan" << endl; }
26      void greet() const { cout << "Greetings from a tiger" << endl;  }
27  };
28
29  int main() {
30    Lion  x;
31    Tiger y;
32
33    array<Animal*,2> tab={&x,&y};
34    for( auto p : tab){
35          p->greet();
36          p->speak();
37    }
38  }
```

5.1) JUSTIFIER tout ce qui est affiché par son exécution

6) (6 pts) héritage multiple : soit le fichier **ex6.cc** dont le code est listé ci-dessous

Ce code **ex6.cc** compile sans warning avec -std=c++11 -Wall. , Son exécution se termine normalement.

```
1   #include <iostream>
2   using namespace std;
3
4   class A {
5   public:
6        A() { cout << "Constructeur A" << endl; }
7        virtual void print() const { cout << "Affichage A" << endl; }
8   };
9
10  class B : virtual public A {
11  public:
12       B() { cout << "Constructeur B" << endl; }
13       void print() const override { cout << "Affichage B" << endl; }
14  };
15
16  class C : virtual public A {
17  public:
18       C() { cout << "Constructeur C" << std::endl; }
19       void print() const override { cout << "Affichage C" << endl; }
20  };
21
22  class D : public B, public C {
23  public:
24       D() { cout << "Constructeur D" << endl; }
25       void print() const override { cout << "Affichage D" << endl; }
26  };
27
28  int main() {
29       D* d1 = new D();
30       A d2 = *d1;
31       d2.print();
32       return 0;
33  }
```

6.1) JUSTIFIER tout ce qui est affiché par son exécution

6.2) Supposons maintenant que la ligne 25 est en commentaire ; qu'en déduisez-vous ?