C++ PoP MT & EL /White lundi 5 juillet 2021

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 _____

Prénom _____

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: **8h45**

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) (10 pts) Surcharge des opérateurs

1.1) Les déclarations C++11 suivantes sont correctes.

```
1   class A
2   {
3   private:
4        int a;
5   public:
6        const A operator+(A const& a) const;
7   };
8
9   class B {
10  private:
11       int b;
12  };
13
14  class C {
15  private:
16       int c;
17  };
18
19  const A operator+(A const& a, A const& b);
20
21  const C operator+(C const& c, C const& b);
```

Choisir pour chaque classe la surcharge d'opérateur qu'elle implémente :

Classe	Entourer la réponse retenue **puis brièvement justifier votre réponse**			
A	Externe	Interne	Les deux	Aucune
B	Externe	Interne	Les deux	Aucune
C	Externe	Interne	Les deux	Aucune

1.2) Dans ce code, les détails de la classe D ne sont pas fournis.

```
1   #include <iostream>
2
3   int main ()
4   {
5      D d1, d2;
6      int e(99);
7
8      d1 += d2;
9
10     std::cout << d1;
11
12     e = 3 + d1;
13
14     e = d1 + 3;
15
16     return 0;
17  }
```

Choisir quelles surcharges d'opérateur la classe D peut définir pour que le code puisse compiler. Si les deux types de surcharges peuvent être définis, il faut indiquer cette réponse.

ligne	Entourer la réponse retenue **puis brièvement justifier votre réponse**		
8	Externe	Interne	Les deux
10	Externe	Interne	Les deux
12	Externe	Interne	Les deux
14	Externe	Interne	Les deux

1.3) Le code ci-dessous présente une erreur de compilation. Décrire l'erreur et expliquer comment la corriger en détaillant les modifications ou ajout de code (préciser le numéro de la ou des lignes modifiées ; indiquer où insérer d'éventuelles nouvelles lignes de code).
Remarque : il n'est pas autorisé de modifier la fonction main().

```
1   #include <iostream>
2
3   class G
4   {
5   public:
6      G(int g): g(g) {};
7      const int operator+(int val) const { return g + val; }
8   private:
9      int g;
10  };
11
12  std::ostream& operator<<(std::ostream& sortie, G const& g)
13  {
14     sortie << g.g;
15     return sortie;
16  }
17
18  int main()
19  {
20     G g1(3);
21
22     std::cout << "Somme: " << g1 + 2;
23     return 0;
24  }
```

2) (6 pts) Erreur sémantique

Ce code compile sans warning en C++11 ; cependant il contient une erreur sémantique.

```
1   #include <iostream>
2   #include <vector>
3
4   typedef std::vector<int> MyList;
5
6   //cette fonction modifie v en multipliant chacun de
7   //ses éléments par la valeur s
8   void mult_s(MyList& v, double s);
9
10  int main()
11  {
12       MyList v={1,2,3};
13       double s(2.);
14
15       for(auto elem : v)
16          std::cout << elem << ' ';
17       std::cout << std::endl;
18
19       mult_s(v,s);
20
21       for(auto elem : v)
22          std::cout << elem << ' ';
23       std::cout << std::endl;
24
25       return 0;
26  }
27
28  void mult_s(MyList& v, double s)
29  {
30       for(auto elem : v)
31         elem *= s ;
32  }
```

Trouver l'erreur sémantique présente dans ce code en précisant les résultats d'affichage des lignes 15-17 et 21-23. Indiquer comment elle peut être corrigée.

3) (9 pts) Polymorphisme : ce code compile sans warning en C++11.

```
1   #include <iostream>
2   using namespace std;
3
4   class A {
5   public:
6        virtual string foo() {
7             return "foo";
8        }
9        virtual string bar() {
10            return "bar";
11       }
12       void print() {
13            cout << foo() << ' ' << bar() << endl;
14       }
15  };
16
17  class B: public A{
18  public:
19       string foo() override {
20            return "boo";
21       }
22
23       void print() {
24            cout << "B ";
25            A::print();
26       }
27  };
28
29  class C: public B {
30  public:
31       string bar() override {
32            return "car";
33       }
34       void print() {
35            cout << "C ";
36            A::print();
37       }
38  };
39
40  int main() {
41       A a;
42       a.print();
43
44       B b;
45       b.print();
46
47       C c;
48       c.print();
49
50       B& br = c;
51       br.print();
52
53       return 0;
54  }
```

Indiquer l'affichage obtenu pour chacune des lignes de code suivantes ;
une justification est exigée pour chaque cas

Ligne 42 :

Ligne 45 :

Ligne 48 :

Ligne 51 :

4) (10 pts) Héritage simple et multiple : ce code compile sans warning en C++11.

```
1   #include <iostream>
2   using namespace std;
3
4   class A
5   {
6       public:
7           A(int x=10, int y=20): x_(x), y_(y) {};
8           ~A() {cout << " A destroyed " << endl; }
9           void affiche() const
10            {cout << "x = " << x_ << ", y = " << y_ << endl;}
11
12      protected:
13          int x_;
14
15      private:
16          int y_;
17  };
18
19  class B : public A
20  {
21      public:
22          void set_x(int x){x_ = x;}
23          int get_x() const {return x_;}
24
25      protected:
26          int x_=30;
27  };
28
29  class C : public virtual A
30  {
31      public:
32          C(): pt_(nullptr) {};
33          ~C() {cout << " C destroyed " << endl; }
34          int* get_pt() const {return pt_;}
35          void set_pt(int z) {pt_ = &x_; *pt_ = z;}
36      protected :
37          int* pt_;
38  };
39
40  class D : public virtual A
41  {
42      public:
43          D(): pt_(nullptr) {};
44          ~D() {cout << " D destroyed " << endl; }
45          void set_pt(int& z) {pt_ = &z;}
46      protected :
47          int* pt_;
48  };
49
50  class E : public D, public C
51  {
52      public :
53          ~E() {cout << " E destroyed " << endl; }
54  };
```

```
55  int main()
56  {
57       { // ce bloc est destiné à limiter la portée de b
58            B b;
59            cout << b.get_x() << endl;     // Q0
60            b.set_x(2);
61            b.affiche();                   // Q1
62            cout << b.get_x() << endl;     // Q2
63       }
64
65       { // ce bloc est destiné à limiter la portée de c
66            C c;
67            //cout << c.x_ << endl;        // Q3
68            //cout << c.y_ << endl;        // Q3
69
70            int reponse = 42;
71            c.set_pt(reponse);
72            reponse = 9;
73            //int* p(c.get_pt());          // Q4
74            //cout << (p ? *p:0) << endl; // Q4
75       }
76
77       { // ce bloc est destiné à limiter la portée de e
78            //int reponse = 42;            // Q5
79            E e ;
80            //e.set_pt(reponse);           // Q5
81       } // Q6
82       return 0;
     }
```

Chaque question est liée à certaines lignes du code indiquées par un commentaire en fin de ligne(s).

Si les instructions d'une question sont commentées (ex : Q3, Q4 et Q5) vous devez indiquer si le code compile quand on enlève ces commentaires et si oui, il faut indiquer le résultat de l'exécution. Si les instructions ne compilent pas, ces lignes restent en commentaire.

Question / Ligne(s)	Affichage obtenu ou description d'erreur de compilation
Q0, ligne 59	
Q1, ligne 61	
Q2, ligne 62	

Q3, lignes 67-68	
Q4, ligne 73-74	
Q5, lignes 78-80	
Q6, ligne 81 **Quel affichage est produit en quittant ce bloc ?**	