ICC MT & EL Test écrit de C++ POP /White vendredi 3 mai 2019

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: **14h15**

A **15h30** retourner les feuilles avec la page 1 face à vous et faire glisser les copies vers les allées pour être ramassées. Attendre en silence le signal pour sortir.

AUCUN appareil électronique n'est autorisé

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

Vous pouvez utiliser un crayon à papier et une gomme

Ce contrôle écrit de C++ permet d'obtenir **40 points** sur un total de 100 points pour le cours complet.

1) (4 pts) Définition et utilisation d'une classe

```
1   #include <iostream>
2   using namespace std;
3
4   class Test
5   {
6   private:
7       int k;
8   public:
9       Test();
10      void reset() const;
11      void print() const;
12  };
13
14  Test::Test():k(-2){}
15
16  void Test::reset() const
17  {
18       k = 2;
19  }
20
21  void Test::print() const
22  {
23       cout << k << endl ;
24  }
25
26  int main()
27  {
28       Test object1;
29
30       object1.reset();
31       object1.print();
32
33       return 0;
34  }
```

Choisir une seule réponse parmi les comportements proposés (standard C++11).

Choix	Comportement observé concernant ce code
A	L'exécution produit l'affichage de 2 puis passe à la ligne
B	L'exécution produit l'affichage de -2 puis passe à la ligne
C	La compilation produit une erreur pour la ligne 18 car k est un attribut **`private`**
D	La compilation produit une erreur pour la ligne 18 à cause du modificateur **`const`**
E	La compilation produit une erreur pour la ligne 23 à cause du modificateur **`const`**

2) (7 pts) classe et static : Le code suivant produit un exécutable (sans warning) en C++11

```
1   #include <iostream>
2   using namespace std;
3
4   class Foo
5   {
6       static int val_;
7   public:
8       int inc()
9       {
10          static int inc = val_++;
11          return ++inc;
12      }
13
14      int val()
15      {
16          return val_;
17      }
18  };
19
20  int Foo::val_ = 17;
21
22  int main()
23  {
24      Foo a, b, c;
25
26      cout << "A | val: " << a.val() << endl;
27      cout << "A | inc: " << a.inc() << endl << endl;
28
29      cout << "B | val: " << b.val() << endl;
30      cout << "B | inc: " << b.inc() << endl << endl;
31
32      cout << "C | val: " << c.val() << endl;
33      cout << "C | inc: " << c.inc() << endl;
34
35      return 0;
36  }
```

Justifier chacun des affichages produits par ce programme dans le tableau ci-dessous :

Ligne	affichage	justification
26		
27		
29		
30		
32		
33		

3) (8 pts) Hiérarchie de classe, méthodes et fonctions

Le code suivant produit un exécutable (sans warning) en C++11

```
1   #include <iostream>
2   using namespace std;
3
4   class TeamMember {
5   protected:
6       string name;
7   public:
8       virtual void memberInfo() const {
9           cout << "Team member name: " << name  << endl;
10      }
11      void setInfo(string name_) {
12          name = name_;
13      }
14  };
15
16  class Racer : public TeamMember {
17  protected:
18      unsigned nbWin;
19  public :
20      void memberInfo() const {
21          cout << "Racer Name:" << name;
22          cout << ", number of victories:" << nbWin << endl;
23      }
24      void setRacerInfo(unsigned numOfWins) {
25          nbWin = numOfWins;
26      }
27  };
28
29  class Swimmer : public TeamMember {
30  protected:
31      double bestTime;
32  public:
33      void memberInfo() const {
34          cout << "Swimmer Name:" << name ;
35          cout << ", best Time:" << bestTime << endl;
36      }
37      void setSwimmerInfo(double time) {
38          bestTime = time;
39      }
40  };
41
42  class Skater : public TeamMember {
43  protected:
44      unsigned nbGold;
45  public:
46      void memberInfo() const {
47          cout << "Skater Name:" << name ;
48          cout << ", number of gold medals:" << nbGold << endl;
49      }
50      void setSkaterInfo(unsigned numOfGold) {
51          nbGold = numOfGold;
52      }
53  };
54
55  void f(TeamMember& v){
56        v.memberInfo();
57  }
58  void g(TeamMember v){
59        v.memberInfo();
60  }
61  void h(TeamMember* v){
62        v->memberInfo();
63  }
64  // SUITE DU CODE SOURCE A LA PAGE SUIVANTE
```

Répondre aux questions indiquées en commentaires en utilisant l'espace libre

```
1   int main() {
2
3   // 3.1) Quel affichage est produit par la ligne 6 ?
4       TeamMember m1;
5       m1.setInfo("Alain");
6       f(m1);
7
8
9
10
11
12  // 3.2) Quel affichage est produit par la ligne 15 ?
13      TeamMember m2;
14      m2.setInfo("Lewis");
15      h(&m2);
16
17
18
19
20
21  // 3.3) Quel affichage est produit par les lignes 25-26 ?
22      Racer m3;
23      m3.setInfo("Sebastian");
24      m3.setRacerInfo(52);
25      m3.memberInfo();
26      f(m3);
27
28
29
30
31
32
33
34
35  // 3.4) Quel affichage est produit par les lignes  39-40 ?
36      Swimmer m4;
37      m4.setInfo("Michael");
38      m4.setSwimmerInfo(47.51);
39      m4.memberInfo();
40      g(m4);
41
42
43
44
45
46
47
48
49  // 3.5) Quel affichage est produit par les lignes  53-54 ?
50      Skater m5;
51      m5.setInfo("Katarina");
52      m5.setSkaterInfo(2);
53      m5.memberInfo();
54      h(&m5);
55
56
57
58
59
60
61
62        return 0;
63  }
```

4) (12 pts) Hiérarchie de classes

Le code visible ci-dessous produit du code objet (sans warning) en C++11

```
1  #include <iostream>
2  using namespace std;
3
4  class A {
5  protected:
6       int x;
7  public:
8       void f(){}
9  };
10
11 class B: public A {
12 protected:
13      int x1;
14 public:
15      void h(A *p_obj1);
16 };
17
18 class C: public B {
19 public:
20      int x2;
21 };
22
23 class D: public A {
24 public:
25      void f(){}
26 };
27
28 class E: public A {
29 public:
30      void g(){}
31 };
```

4.1) Dessiner la hiérarchie de classes correspondant aux déclarations ci-contre ; respecter le sens des flèches pour exprimer la relation « est-un »

4.2) On ajoute la fonction main() suivante après la ligne 31. Compléter la colonne indiquant si il y a une erreur (oui) ou pas (non) pour les lignes 34, 37, 40, 43 :

```
32 int main(){
33     B obj1;
34     obj1.f();
35
36
37     cin >> obj1.x;
38
39
40     cin >> obj1.x1;
41
42
43     cin >> obj1.x2;
44
45
46     return 0;
47 }
```

Erreur ?
……
……
……
……

(suite) Justifier brièvement ci-dessous :

Ligne 34 :

Ligne 37 :

Ligne 40 :

Ligne 43 :

4.3) ***Cette question est indépendante de la précédente***. Ici on complète la définition de la hiérarchie de classes en ajoutant la définition de la méthode **h()** de la classe **B** après la ligne 31. Compléter la colonne indiquant si il y a une erreur (oui) ou pas (non) pour les lignes 38, 41, 44:

32	`void B::h(A *p_obj1){`	Erreur ?
33		
34	`C *p_obj2( new C);`	……non……
35	`D *p_obj3( new D);`	……non……
36		
37		
38	`cin >> p_obj1->x ;`	……
39		
40		
41	`cin >> p_obj2->x ;`	……
42		
43		
44	`cin >> p_obj3->x ;`	……
45		
46	`}`	

(suite) Justifier brièvement ci-dessous :

Ligne 38:

Ligne 41:

Ligne 44 :

5) (9 pts) Constructeur et destructeur,

Le code visible ci-dessous produit un exécutable (sans warning) en C++11

```
1   #include <iostream>
2   using namespace std;
3
4   int i(7);
5
6   class A{
7   public:
8      A(){
9         cout << "Pomme" <<endl;
10        i = 5;
11     }
12     ~A(){
13        cout << "Banane" <<endl;
14        i = 10;
15     }
16  };
17
18  class B: public A {
19  public:
20     B(){
21        cout << "Ananas" <<endl;
22        i=2;
23     }
24     ~B(){
25        cout << "Poire" <<endl;
26        i=4;
27     }
28  };
29
30  int foo(){
31     if(i==7){
32        B obj;
33     }
34     return i;
35  }
36
37  int main(){
38     cout << foo() << endl;
39     return 0;
40  }
```

Déterminer le nombre de lignes affiché par ce programme. S'il y a plus d'une ligne, respecter l'ordre d'affichage. Dans tous les cas, justifier le contenu affiché par chaque ligne :

Ordre	Contenu de la ligne affichée	Justification
1		
2		
3		
4		
5		