**TEST Sem2 MT & EL 03.05.2019 EPFL / R. Boulic**

**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

**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). [All]

| 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 :[Wh]

| **Ligne** | affichage | justification |
|---|---|---|
| **26** | **17** | La valeur initiale de la variable de classe **val_** est **17** ; initialisée à la ligne 20 |
| **27** | **18** | Initialise ***une seule fois*** la variable **static inc** de la méthode **inc()** avec la valeur 17 de **val_** avant incrémentation. Ensuite pré-incrémente la variable **inc** et renvoie sa nouvelle valeur **18 .** |
| **29** | **18** | La post-incrémentation effectuée à la ligne 10 a modifié la variable de classe **val_** qui vaut **18** |
| **30** | **19** | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **18** et le renvoie de **19** |
| **32** | **18** | la variable de classe **val_** vaut toujours **18** car elle n'a été modifiée qu'***une seule fois*** |
| **33** | **20** | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **19** et le renvoie de **20** |

**TEST Sem2 MT & EL 03.05.2019 EPFL / R. Boulic**
**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

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

| **Ligne** | affichage | justification |
|---|---|---|
| **26** | 18 | La valeur initiale de la variable de classe **val_** est 18 ; initialisée à la ligne 20 |
| **27** | 19 | Initialise ***une seule fois*** la variable static **inc** de la méthode **inc()** avec la valeur 18 de **val_** avant incrémentation. Ensuite pré-incrémente la variable **inc** et renvoie sa nouvelle valeur **19 .** |
| **29** | 19 | La post-incrémentation effectuée à la ligne 10 a modifié la variable de classe **val_** qui vaut **19** |
| **30** | 20 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **19** et le renvoie de **20** |
| **32** | 19 | La variable de classe **val_** vaut toujours **19** car elle n'a été modifiée qu'***une seule fois*** |
| **33** | 21 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **20** et le renvoie de **21** |

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

| **Ligne** | affichage | justification |
|---|---|---|
| **26** | 19 | La valeur initiale de la variable de classe **val_** est **19** ; initialisée à la ligne 20 |
| **27** | 20 | Initialise ***une seule fois*** la variable static **inc** de la méthode **inc()** avec la valeur 19 de **val_** avant incrémentation. Ensuite pré-incrémente la variable **inc** et renvoie sa nouvelle valeur **20.** |
| **29** | 20 | La post-incrémentation effectuée à la ligne 10 a modifié la variable de classe **val_** qui vaut **20** |
| **30** | 21 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **20** et le renvoie de **21** |
| **32** | 20 | La variable de classe **val_** vaut toujours **20** car elle n'a été modifiée qu'***une seule fois*** |
| **33** | 22 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **21** et le renvoie de **22** |

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

| **Ligne** | affichage | justification |
|---|---|---|
| **26** | 16 | La valeur initiale de la variable de classe **val_** est **16** ; initialisée à la ligne 20 |
| **27** | 17 | Initialise ***une seule fois*** la variable static **inc** de la méthode **inc()** avec la valeur 16 de **val_** avant incrémentation. Ensuite pré-incrémente la variable **inc** et renvoie sa nouvelle valeur **17 .** |
| **29** | 17 | La post-incrémentation effectuée à la ligne 10 a modifié la variable de classe **val_** qui vaut **17** |
| **30** | 18 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **17** et le renvoie de **18** |
| **32** | 17 | La variable de classe **val_** vaut toujours **17** car elle n'a été modifiée qu'***une seule fois*** |
| **33** | 19 | Plus d'initialisation de **inc** ; seulement sa préincrémentation sur la valeur **18** et le renvoie de **19** |

**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

### 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() { [Wh]
2
3   // 3.1) Quel affichage est produit par la ligne 6 ?
4       TeamMember m1;
5       m1.setInfo("Alain");
6       f(m1);
7
8   Team member name : Alain
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  Team member name : Lewis
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  Racer Name:Sebastian, number of victories:52
29  Racer Name:Sebastian, number of victories:52
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  Swimmer Name:Michael, best Time :47.51
44  Team member name: Michael
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  Skater Name:Katarina, number of gold medals:2
57  Skater Name:Katarina, number of gold medals:2
58
59
60        return 0;
61  }
62
63
```

```
1   int main() { [B]
2
3   // 3.1) Quel affichage est produit par la ligne 6 ?
4       TeamMember m1;
5       m1.setInfo("Alain");
6       h(&m1);
7
8   Team member name : Alain
9
10
11
12  // 3.2) Quel affichage est produit par la ligne 15 ?
13      TeamMember m2;
14      m2.setInfo("Lewis");
15      f(m2);
16
17  Team member name : Lewis
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      g(m3);
27
28
29  Racer Name:Sebastian, number of victories:52
30  Team member name : Sebastian
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      f(m4);
41
42
43  Swimmer Name:Michael, best Time :47.51
44  Swimmer Name:Michael, best Time :47.51
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  Skater Name:Katarina, number of gold medals:2
58  Skater Name:Katarina, number of gold medals:2
59
60
61
62        return 0;
63  }
```

```
1   int main() { [G]
2
3   // 3.1) Quel affichage est produit par la ligne 6 ?
4       TeamMember m1;
5       m1.setInfo("Alain");
6       f(m1);
7
8   Team member name : Alain
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  Team member name : Lewis
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      h(&m3);
27
28
29  Racer Name:Sebastian, number of victories:52
30  Racer Name:Sebastian, number of victories:52
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      f(m4);
41
42
43  Swimmer Name:Michael, best Time :47.51
44  Swimmer Name:Michael, best Time :47.51
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      g(m5);
55
56
57  Skater Name:Katarina, number of gold medals:2
58  Team member name : Katarina
59
60
61
62        return 0;
63  }
```

```
1   int main() { [Y]
2
3   // 3.1) Quel affichage est produit par la ligne 6 ?
4       TeamMember m1;
5       m1.setInfo("Alain");
6       h(&m1);
7
8   Team member name : Alain
9
10
11
12  // 3.2) Quel affichage est produit par la ligne 15 ?
13      TeamMember m2;
14      m2.setInfo("Lewis");
15      f(m2);
16
17  Team member name : Lewis
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      g(m3);
27
28
29  Racer Name:Sebastian, number of victories:52
30  Team member name : Sebastian
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      h(&m4);
41
42
43  Swimmer Name:Michael, best Time :47.51
44  Swimmer Name:Michael, best Time :47.51
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      f(m5);
55
56
57  Skater Name:Katarina, number of gold medals:2
58  Skater Name:Katarina, number of gold medals:2
59
60
61
62        return 0;
63  }
```

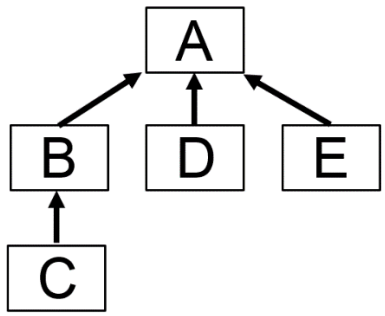
**W=blanc, B=bleu, G=vert, Y=jaune, All = même réponse pour tous**

**4) (12 pts) Hiérarchie de classes**

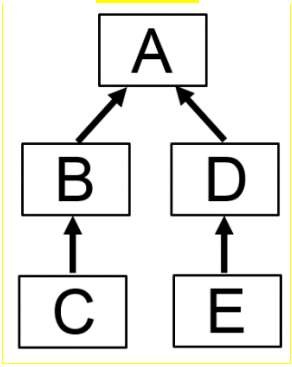
Le code visible ci-dessous produit du code objet (sans warning) en C++11

```
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 » [Wh] + [G]




[B] + [Y]




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 : [All]

```
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 ? |
|---|
| ……non…… |
| ……oui… |
| ……oui… |
| ……oui… |

(suite) Justifier brièvement ci-dessous : (pour 4.2 et 4.3 revoir le MOOC semaine 4 : seconde video avec ses quizzes + p 39 du polycopié)

Ligne 34 : **OK pas d'erreur** car f() est une méthode publique de A qui est héritée par B

Ligne 37 : **erreur** car **x** est un attribut **protégé** de A ⇔ on ne se trouve pas dans la portée de la classe A ou B

Ligne 40 : **erreur** car **x1** est un attribut **protégé** de B ⇔ on ne se trouve pas dans la portée de la classe B

Ligne 43 : **erreur** car **x2** est seulement défini dans une classe dérivée de B ; une instance de B ne connaît pas cet attribut.

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:

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

(suite) Justifier brièvement ci-dessous : le texte surligné en bleu est un complément d'explication qui n'est pas exigé pour la réponse

Ligne 38: **erreur** car **x** est un attribut **protégé** de A et **p_obj1** est un pointeur *sur un objet de la classe parente A ;* or une expression sur un objet de la classe A n'appartient pas à la portée de B. On ne peut accéder à **x** qu'à travers un objet dérivé appartenant à la sous-classe B. Par exemple on a le droit d'écrire : cin >> this->x.

Ligne 41: **OK pas d'erreur** car le pointeur **p_obj2** est celui d'une instance de C dérivée en « public » de la classe B ; de ce fait elle fait partie de la portée de B. De plus l'attribut **x** est **protégé** au niveau de A ce qui autorise son accès pour les classes dérivées.

Ligne 44 : **erreur** car la classe D n'est pas dérivée de B mais de A ; de ce fait l'accès à l'attribut **protégé x** de A n'est pas autorisé car cette expression n'appartient pas à la portée de B.

**5) (9 pts) Constructeur et destructeur,**

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

```
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 | Pomme | La variable globale i vaut 7 lors du test ligne 31 qui est vrai et produit la déclaration de obj à la ligne 32 qui lui-même produit l'appel des constructeurs en commençant par celui de la classe A (affichage ligne13) |
| 2 | Ananas | Ensuite vient l'appel du constructeur de la classe dérivée B qui produit l'affichage de la ligne 21 |
| 3 | Poire | Obj finit d'exister à la ligne 33 ; le destructeur de B est appelé en premier et produit l'affichage de la ligne 25 |
| 4 | Banane | Le dernier destructeur appelé est celui de A, ligne 13 |
| 5 | 10 | La dernière instruction modifiant la variable globale avecla valeur 10 est dans le destructeur de A en ligne 14. C'est cette valeur qui est renvoyée par foo() et affichée |

**5) (9 pts) Constructeur et destructeur,**

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

```
1   #include <iostream>
2   using namespace std;
3
4   int i(5);
5
6   class A{
7   public:
8      A(){
9         cout << "Ananas" <<endl;
10         i = 10;
11     }
12     ~A(){
13        cout << "Poire" <<endl;
14         i = 7;
15     }
16  };
17
18  class B: public A {
19  public:
20     B(){
21        cout << "Pomme" <<endl;
22         i=4;
23     }
24     ~B(){
25        cout << "Banane" <<endl;
26         i=2;
27     }
28  };
29
30  int foo(){
31     if(i==5){
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 | Ananas | La variable globale i vaut 5 lors du test ligne 31 qui est vrai et produit la déclaration de obj à la ligne 32 qui lui-même produit l'appel des constructeurs en commençant par celui de la classe A (affichage ligne13) |
| 2 | Pomme | Ensuite vient l'appel du constructeur de la classe dérivée B qui produit l'affichage de la ligne 21 |
| 3 | Banane | Obj finit d'exister à la ligne 33 ; le destructeur de B est appelé en premier et produit l'affichage de la ligne 25 |
| 4 | Poire | Le dernier destructeur appelé est celui de A, ligne 13 |
| 5 | 7 | La dernière instruction modifiant la variable globale avec la valeur 7 est dans le destructeur de A en ligne 14. C'est cette valeur qui est renvoyée par foo() et affichée |

**5) (9 pts) Constructeur et destructeur,**

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

```
1   #include <iostream>
2   using namespace std;
3
4   int i(4);
5
6   class A{
7   public:
8      A(){
9         cout << "Pomme" <<endl;
10         i = 7;
11     }
12     ~A(){
13        cout << "Poire" <<endl;
14         i = 5;
15     }
16  };
17
18  class B: public A {
19  public:
20     B(){
21        cout << "Ananas" <<endl;
22         i=10;
23     }
24     ~B(){
25        cout << "Banane" <<endl;
26         i=2;
27     }
28  };
29
30  int foo(){
31     if(i==4){
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 | Pomme | La variable globale i vaut 4 lors du test ligne 31 qui est vrai et produit la déclaration de obj à la ligne 32 qui lui-même produit l'appel des constructeurs en commençant par celui de la classe A (affichage ligne13) |
| 2 | Ananas | Ensuite vient l'appel du constructeur de la classe dérivée B qui produit l'affichage de la ligne 21 |
| 3 | Banane | Obj finit d'exister à la ligne 33 ; le destructeur de B est appelé en premier et produit l'affichage de la ligne 25 |
| 4 | Poire | Le dernier destructeur appelé est celui de A, ligne 13 |
| 5 | 5 | La dernière instruction modifiant la variable globale avecla valeur 5 est dans le destructeur de A en ligne 14. C'est cette valeur qui est renvoyée par foo() et affichée |

**5) (9 pts) Constructeur et destructeur,**

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

```
1  #include <iostream>
2  using namespace std;
3
4  int i(10);
5
6  class A{
7  public:
8     A(){
9        cout << "Ananas" <<endl;
10       i = 7;
11    }
12    ~A(){
13       cout << "Banane" <<endl;
14       i = 5;
15    }
16 };
17
18 class B: public A {
19 public:
20    B(){
21       cout << "Pomme" <<endl;
22       i=2;
23    }
24    ~B(){
25       cout << "Poire" <<endl;
26       i=4;
27    }
28 };
29
30 int foo(){
31    if(i==10){
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 | Ananas | La variable globale i vaut 10 lors du test ligne 31 qui est vrai et produit la déclaration de obj à la ligne 32 qui lui-même produit l'appel des constructeurs en commençant par celui de la classe A (affichage ligne13) |
| 2 | Pomme | Ensuite vient l'appel du constructeur de la classe dérivée B qui produit l'affichage de la ligne 21 |
| 3 | Poire | Obj finit d'exister à la ligne 33 ; le destructeur de B est appelé en premier et produit l'affichage de la ligne 25 |
| 4 | Banane | Le dernier destructeur appelé est celui de A, ligne 13 |
| 5 | 5 | La dernière instruction modifiant la variable globale avec la valeur 5 est dans le destructeur de A en ligne 14. C'est cette valeur qui est renvoyée par foo() et affichée |