QCM - Constructeurs/Destructeurs

A. Choisissez les affirmations correctes :

1. Tant que le programmeur n'a pas fait explicitement appel au constructeur, il n'y a pas d'instance en mémoire.
2. Dès que l'on déclare un objet, il y a automatiquement appel à un constructeur (ou erreur de compilation s'il n'y a pas de constructeur possible).
3. Un constructeur est forcément appelé par le compilateur.
4. C'est au programmeur de faire appel au constructeur.

B. Parmi les extraits de code suivants, lesquels contiennent des appels corrects à des constructeurs (que l'on suppose exister ; ce n'est pas ici le but de la question).

1. Rectangle r;
2. Rectangle r = rectangle(1.1, 2.2);
3. Rectangle r = (1.1, 2.2);
   new Rectangle r(1.1, 2.2);
4. Rectangle r(1.1, 2.2);
5. Rectangle r = new Rectangle(1.1, 2.2);
6. Rectangle r; r(1.1, 2.2);

C. La classe OursEnPeluche a deux attributs :

(1) un **pelage** qui est une instance d'une **classe Materiau,** laquelle a un constructeur à deux paramètres : une couleur et un textile ;
(2) un **prix** de type **double**.

Lequel des constructeurs suivants est correct pour la classe OursEnPeluche ?

1. OursEnPeluche(Textile tissu, Couleur c,
   double somme)
   {}

2. OursEnPeluche(Textile tissu, Couleur c,
   double somme)
   : Materiau(tissu, c), prix(somme)
   {}

3. OursEnPeluche(Textile tissu, Couleur c,
   double somme)
   : pelage(tissu, c), prix(somme)
   {}

4. OursEnPeluche(Textile tissu, Couleur c,
double somme)
{ pelage = Materiau(tissu, c);
prix = somme;
}

5. OursEnPeluche(Textile tissu, Couleur c,
double somme)
: Materiau pelage(tissu, c),
double prix(somme)
{}

D. Pour une classe OursEnPeluche, laquelle des lignes de codes suivantes ne peut en aucun cas être une initialisation (i.e. un appel à un constructeur) ?

1. OursEnPeluche mon_doudou;
2. OursEnPeluche mon_doudou();
3. OursEnPeluche mon_doudou(brun);
4. OursEnPeluche mon_doudou(laine, rose, 12.35);

E. Comment le compilateur interprète-t-il Rectangle r();

1. comme une initialisation par défaut d'une instance r de la classe Rectangle
2. comme un prototype d'une fonction sans paramètre retournant un Rectangle
3. comme un appel de fonction
4. comme un constructeur de Rectangle

F. Etant donné le code suivant, cochez toutes les assertions correctes.

```
class Bagel
{
public:
Bagel(bool beurre = false)
: avecBeurre(beurre) {}

void affiche() const {
cout << "Un bagel";

if (avecBeurre) {
cout << " au beurre";
}
cout << endl;
}

private:
bool avecBeurre;
};
```

1. Bagel b; b.affiche();

affiche: Un bagel car l'attribut a été initialisé à false par le constructeur par défaut.

2. Bagel b(true); b.affiche();

affiche: Un bagel au beurre

3. Bagel bagel;

fait appel au constructeur par défaut par défaut

4. Bagel bagel;

ne compile pas car il n'y a pas de constructeur par défaut.

G. Cochez tous les codes suivants qui compilent et pour lesquels Balle b; construira une balle de rayon 2.

1. class Balle

```
{
public:
Balle(int d = 2) : rayon(d) { }
private:
int rayon;
};
```

2. class Balle

```
{
public:
Balle(int d) : rayon(d) { }
Balle() : Balle(2) {}
private:
int rayon;
};
```

3. class Balle

```
{
public:
Balle(int d) : rayon(d) {}
Balle() : Balle(3) {}
private:
int rayon = 2;
};
```

4. class Balle

```
{
private:
int rayon = 2;
};
```

H. Que faut il rajouter dans la classe A pour que le code suivant compile ?

```
class A {
// ... des attributs ...
};
int main()
{ A a1;
A a2(a1);
// ...
}
```

1. rien du tout, cela compile en l'état
2. le constructeur de copie
3. le constructeur par défaut

4. le constructeur de copie et le constructeur par défaut

I. Cochez les codes corrects parmi les suivants :

1. class Monde

```
{
~Monde(){ }
};
```

2. class Monde

```
{
~Monde(bool tout_detruire = true) {
//...
}
};
```

3. class Monde

```
{ ~Monde(bool tout_detruire ) {
//...
}
~Monde(){}
};
```

J. Soit le code suivant (on suppose que toutes les inclusions nécessaires sont faites) :

```
class Aventure {
public:
~Aventure() {
cout << "fin d'une aventure" << endl;
}
};
int main()
{
Aventure a1;
{
Aventure a2;
Aventure a3;
}
return 0;
}
```

Combien de fois affiche t-il le message « fin d'une aventure » ?

1. aucune fois
2. une fois
3. deux fois
4. trois fois

K. Cochez toutes les assertions correctes parmi les suivantes :

1. Le mot réservé static est utilisé pour indiquer qu'un attribut ne change pas de valeur
2. Appliqué à un attribut d'une classe C, le mot réservé static indique que ce dernier est partagé par toutes les instances de C
3. Un attribut statique att d'une classe C peut-être accédé par la notation C::att
4. Un attribut statique est forcément privé

L. Soit une classe C disposant d'un attribut statique att de type int. Cochez toutes les assertions correctes parmi les suivantes :

1. La ligne : int C::att(15), placée en dehors de C, permet d'initialiser att à 15
2. La ligne : int C::att(15), placée dans le constructeur de C, permet d'initialiser att à 15
3. La ligne : int att(15), placée en dehors de C, permet d'initialiser att à 15
4. La ligne : int C.att(15), placée en dehors de C, permet d'initialiser att à 15

QCM - Héritage

A. Cochez les assertions correctes parmi les suivantes :

1. L'héritage permet d'éviter des redondances dans le contenu de classes
2. L'héritage permet d'expliciter les liens sémantiques qui existent entres des classes
3. L'héritage permet de modéliser la relation «A-UN» entre classes
4. L'héritage permet de modéliser la relation «EST-UN» entre classes

B. Cochez toutes les assertions suivantes à propos du code suivant :

```
class A
{
public:
int aToi;

private:
int aMoi;
};
class B : public A
{
private:
void m() {
```

```
// ...
}
};
```

1. la méthode m ne peut utiliser l'attribut aMoi car il est privé

La classe B dispose bien de tous les attributs hérités (y compris l'attribut privé). Elle ne peut cependant y accéder directement.
Le fait que la méthode m soit privée dans B n'a aucune incidence sur ce qu'elle a le droit d'accéder.

2. la méthode m ne peut utiliser l'attribut aToi car elle est privée

3. la méthode m ne peut utiliser l'attribut aMoi car B n'a pas d'attribut de ce nom

4. la méthode m ne peut utiliser l'attribut aToi car il est n'est pas défini dans B

C. La classe Banquier contient une méthode protégée void ouvrirCoffre().

```
class Tresorier : public Banquier {
public:
void faireOuvrir(Banquier b)
{ b.ouvrirCoffre(); }
void faireOuvrir(Tresorier t)
{ t.ouvrirCoffre(); }
};
class Voleur {
public:
void faireOuvrir(Banquier b)
{ b.ouvrirCoffre(); }
void faireOuvrir(Tresorier t)
{ t.ouvrirCoffre(); }
};
```

1. Le code ne compile pas car le trésorier n'a pas le droit de faire ouvrir le coffre à un trésorier.

2. Le code ne compile pas car le trésorier n'a pas le droit de faire ouvrir le coffre à un banquier.

3. Le code ne compile pas car le voleur n'a pas le droit de faire ouvrir le coffre à un trésorier.

4. Le code ne compile pas car le voleur n'a pas le droit de faire ouvrir le coffre à un banquier.