QCM - Classes

Question 1. Quel est l'affichage du programme suivant ?

```
class Compteur {
    int n = 0;
public:
    void increment() { n++; }
    int valeur() const { return n; }
};

int main() {
    Compteur c;
    c.increment();
    c.increment();
    std::cout << c.valeur();
}
```

A) 0
B) 1
C) 2
D) Erreur de compilation

Question 2. Que se passe-t-il si l'on tente d'accéder directement à un attribut privé d'une classe depuis l'extérieur ?

A) L'attribut est initialisé à zéro
B) Le programme compile mais produit un comportement indéfini
C) Le compilateur génère une erreur de compilation
D) L'accès est autorisé en lecture seule

Question 3. Quel est le rôle du constructeur par défaut ?

A) Il détruit l'objet à la fin de sa durée de vie
B) Il initialise l'objet sans nécessiter d'argument
C) Il empêche la copie de l'objet
D) Il convertit implicitement un type en un autre

Question 4. Quelle syntaxe utilise-t-on pour la liste d'initialisation d'un constructeur ?

A) `constructeur() { x = val; }`
B) `constructeur() -> x(val) {}`
C) `constructeur() : x(val) {}`
D) `constructeur() = x(val) {}`

Question 5. Que signifie le mot-clé `const` placé après la signature d'une méthode ?

A) La méthode ne peut pas être appelée
B) La méthode ne modifie pas l'état de l'objet
C) La méthode retourne une valeur constante

D) La méthode ne prend aucun paramètre

Question 6. Quel est le rôle du destructeur en C++ ?

A) Il initialise les attributs de l'objet
B) Il est appelé automatiquement quand l'objet est détruit pour libérer des ressources
C) Il empêche la création de nouvelles instances
D) Il copie l'objet dans un autre emplacement mémoire

Question 7. Que représente le pointeur `this` dans une méthode de classe ?

A) Un pointeur vers la classe de base
B) Un pointeur vers le premier attribut de la classe
C) Un pointeur vers l'objet courant sur lequel la méthode est appelée
D) Un pointeur vers le constructeur de la classe

Question 8. Quel opérateur surcharge-t-on typiquement pour afficher un objet avec `std::cout` ?

A) `operator>>`
B) `operator()`
C) `operator<<`
D) `operator[]`

Question 9. Qu'est-ce qu'un espace de noms (`namespace`) en C++ ?

A) Un type de classe abstraite
B) Un mécanisme pour regrouper des symboles sous un préfixe commun afin d'éviter les conflits de noms
C) Un fichier d'en-tête spécial
D) Un alias pour un type de donnée

Question 10. Considérez le code suivant. Quelle ligne provoque une erreur ?

```
class A {
    int x;
public:
    int y;
};

int main() {
    A a;
    a.x = 1; // ligne 1
    a.y = 2; // ligne 2
}
```

A) Ligne 1 uniquement
B) Ligne 2 uniquement
C) Les deux lignes
D) Aucune erreur

Question 11. Quel est l'effet du mot-clé `explicit` sur un constructeur à un argument ?

A) Il rend le constructeur privé
B) Il empêche les conversions implicites via ce constructeur
C) Il force l'appel du constructeur par défaut
D) Il interdit l'utilisation de la liste d'initialisation

Question 12. Pourquoi la liste d'initialisation est-elle obligatoire pour les attributs `const` et les références ?

A) Parce que le compilateur l'exige pour des raisons de performance
B) Parce que ces membres doivent être initialisés immédiatement et ne peuvent pas être assignés après construction
C) Parce que `const` et les références ne sont pas des types valides dans le corps du constructeur
D) Parce que la liste d'initialisation est toujours obligatoire en C++

Question 13. Considérez le code suivant. Quel est le résultat ?

```
struct vec3 {
    float x, y, z;
    vec3& operator+=(vec3 const& v) {
        x += v.x; y += v.y; z += v.z;
        return *this;
    }
};

vec3 operator+(vec3 a, vec3 const& b) {
    a += b;
    return a;
}

int main() {
    vec3 a{1,2,3};
    vec3 b{4,5,6};
    vec3 c = a + b;
    // Quelles sont les valeurs de a.x, a.y, a.z ?
}
```

A) `a` vaut `{5, 7, 9}`
B) `a` vaut `{1, 2, 3}`
C) `a` vaut `{4, 5, 6}`
D) Le code ne compile pas

Question 14. Quel est le niveau d'accès du membre `x` dans la classe dérivée `Circle` ?

```
class Shape {
protected:
    float x, y;
};

class Circle : public Shape {
public:
    float get_x() const { return x; }
};
```

A) `x` est public dans `Circle`
B) `x` est protected dans `Circle` et accessible depuis ses méthodes
C) `x` est privé dans `Circle`
D) `x` n'est pas accessible dans `Circle`

Question 15. Que signifie `= 0` à la fin d'une déclaration de méthode virtuelle ?

```
virtual float area() const = 0;
```

A) La méthode retourne toujours 0
B) La méthode est une fonction virtuelle pure et la classe est abstraite
C) La méthode est supprimée
D) La méthode a une implémentation par défaut qui retourne 0

Question 16. Pourquoi faut-il un destructeur virtuel dans une hiérarchie polymorphique ?

A) Pour empêcher la création d'objets de la classe de base
B) Pour que le destructeur de la classe dérivée soit correctement appelé lors d'un `delete` via un pointeur de base
C) Pour permettre la surcharge du destructeur
D) Pour que le constructeur de la classe dérivée fonctionne

Question 17. Qu'est-ce que le problème du *slicing* en C++ ?

A) Une erreur de compilation lors de l'utilisation de `virtual`
B) La perte de la partie dérivée d'un objet lorsqu'il est copié dans un objet de type base
C) Un dépassement de tableau
D) Un conflit de noms entre classes

Question 18. Quelle est la particularité d'un attribut `static` dans une classe ?

A) Il est détruit automatiquement à chaque fin de fonction
B) Il est unique et partagé par toutes les instances de la classe
C) Il ne peut être accédé que depuis le constructeur
D) Il doit être initialisé dans chaque constructeur

Question 19. Pourquoi `using namespace std;` est-il déconseillé dans un fichier `.hpp` ?

A) Parce que cela provoque une erreur de compilation
B) Parce que cela pollue l'espace de noms de tous les fichiers qui incluent cet en-tête
C) Parce que `std` n'est pas un namespace valide dans un en-tête
D) Parce que les en-têtes ne supportent pas les directives `using`

Question 20. Considérez le code suivant. Quelle version de `operator[]` est appelée sur l'objet `b` ?

```
class vec3 {
public:
    float x, y, z;
    float& operator[](int i) { return (&x)[i]; }
    float const& operator[](int i) const { return (&x)[i]; }
};

int main() {
    const vec3 b{1,2,3};
    float val = b[0];
}
```

A) La version non `const`
B) La version `const`
C) Les deux versions sont appelées
D) Aucune version n'est appelée, le code ne compile pas

Question 21. Considérez le code suivant. Qu'affiche ce programme ?

```cpp
#include <iostream>

class Shape {
public:
    virtual float area() const { return 0.0f; }
    virtual ~Shape() = default;
};

class Circle : public Shape {
public:
    float r;
    Circle(float r_) : r(r_) {}
    float area() const override { return 3.14f * r * r; }
};

void print_area(Shape s) {
    std::cout << s.area() << std::endl;
}

int main() {
    Circle c(1.0f);
    print_area(c);
}
```

A) `3.14`
B) `0`
C) Erreur de compilation
D) Comportement indéfini

Question 22. Considérez le code suivant. Que se passe-t-il à la compilation ?

```cpp
class Shape {
public:
    virtual float area() const = 0;
    virtual ~Shape() = default;
};

class Rectangle : public Shape {
public:
    float w, h;
    Rectangle(float w_, float h_) : w(w_), h(h_) {}
    float area() { return w * h; } // sans const ni override
};

int main() {
    Rectangle r(3.0f, 4.0f);
}
```

A) Le code compile et `Rectangle::area()` redéfinit correctement `Shape::area()`
B) Erreur de compilation : `Rectangle` est abstraite car `area() const` n'est pas redéfinie
C) Le code compile mais `area()` n'est jamais appelée
D) Erreur de compilation : `override` est obligatoire

Question 23. Considérez le code suivant. Qu'affiche ce programme ?

```cpp
#include <iostream>

class Base {
public:
    static int count;
    Base() { ++count; }
    virtual ~Base() { --count; }
};

int Base::count = 0;

class Derived : public Base {
public:
    Derived() { ++count; }
    ~Derived() override { --count; }
};

int main() {
    {
        Derived d;
    }
    std::cout << Base::count << std::endl;
}
```

A) `0`
B) `1`
C) `-1`
D) `2`

Question 24. Considérez le code suivant. Quelle affirmation est correcte ?

```cpp
class A {
public:
    void f() { std::cout << "A"; }
};

class B : public A {
public:
    void f() { std::cout << "B"; }
};

int main() {
    B b;
    A* ptr = &b;
    ptr->f();
}
```

A) Affiche `B` car `ptr` pointe vers un objet de type `B`
B) Affiche `A` car `f()` n'est pas virtuelle, donc la résolution est statique
C) Erreur de compilation : on ne peut pas affecter un `B*` à un `A*`
D) Comportement indéfini

Question 25. Considérez le code suivant. Quelle est la valeur de `n` ?

```
class Object {
public:
    Object() : id(next_id++) {}
    int get_id() const { return id; }
    static int get_next_id() { return next_id; }
private:
    int id;
    static int next_id;
};

int Object::next_id = 0;

int main() {
    Object a;
    Object b;
    Object c;
    int n = Object::get_next_id() + a.get_id();
}
```

A) `n = 3`
B) `n = 4`
C) `n = 0`
D) `n = 6`