QCM - Introduction au C++

Exercices - Chapitre 1 - Solutions [CSC-43043-EP] 2026

Solutions

Question 1. Réponse : **B)**

1

L'opérateur `%` (modulo) retourne le reste de la division entière. `10 / 3 = 3` avec un reste de `1`, donc `10 % 3` vaut `1`. Cet opérateur ne fonctionne qu'avec des types entiers en C++.

Question 2. Réponse : **A)**

```
std::string s = "hello";
```

En C++, les chaînes de caractères de la bibliothèque standard se déclarent avec `std::string` et s'initialisent avec des guillemets doubles `"..."`. Les guillemets simples `'...'` sont réservés aux caractères individuels (`char`). L'option C utilise la syntaxe des tableaux C, qui n'est pas valide sous cette forme.

Question 3. Réponse : **C)**

Compilé en code machine natif

Le C++ est un langage compilé qui produit du code machine natif directement exécuté par le processeur. Contrairement à Python (interprété) ou Java (exécuté sur la JVM), il n'y a pas de couche intermédiaire. Cette compilation directe garantit des performances élevées, ce qui est l'un des principaux atouts du C++.

Question 4. Réponse : **C)**

```
0 1
```

L'opérateur `&&` (ET logique) retourne `true` uniquement si les deux opérandes sont `true` : ici `true && false` donne `false`. L'opérateur `||` (OU logique) retourne `true` si au moins un opérande est `true` : ici `true || false` donne `true`. En C++, `std::cout` affiche les booléens sous forme numérique par défaut : `0` pour `false` et `1` pour `true`.

Question 5. Réponse : **C)**

```
#include <iostream>
```

La directive `#include <iostream>` permet d'inclure la bibliothèque standard d'entrées/sorties en C++. Elle donne accès à `std::cout` pour l'affichage et `std::cin` pour la lecture. La syntaxe `#import` n'existe pas en C++ standard, et `<stdio>` est l'en-tête C (on utilise `<cstdio>` en C++).

Question 6. Réponse : **B)**

```
int main()
```

Tout programme C++ doit posséder une fonction `main` dont la signature est `int main()`. Cette fonction est le point d'entrée du programme et renvoie un entier au système d'exploitation (0 en cas de succès). La signature `void main()` n'est pas conforme au standard C++.

Question 7. Réponse : **B)**

`<<`

L'opérateur `<<` est l'opérateur d'insertion qui permet d'envoyer des données dans le flux de sortie `std::cout`. L'opérateur `>>` est utilisé avec `std::cin` pour la lecture. L'opérateur `->` sert à accéder aux membres d'un pointeur, et `::` est l'opérateur de résolution de portée.

Question 8. Réponse : **C)**

4 octets

Sur la plupart des machines modernes, un `int` est encodé sur 4 octets (32 bits). Cela permet de représenter des entiers signés dans l'intervalle d'environ -2 milliards à +2 milliards. Le type `char` occupe 1 octet, et le type `double` occupe 8 octets.

Question 9. Réponse : **C)**

`auto`

Le mot-clé `auto`, introduit en C++11, permet au compilateur de déduire automatiquement le type d'une variable à partir de son initialisation. Par exemple, `auto a = 5;` déduit le type `int`. Il est surtout utile pour des types complexes ou des fonctions génériques, et il est recommandé d'utiliser le type explicite pour les types simples.

Question 10. Réponse : **B)**

`6 5`

L'opérateur post-incrémentation `x++` retourne la valeur de `x` avant de l'incrémenter. Ainsi, `y` reçoit l'ancienne valeur `5`, puis `x` passe à `6`. Avec la pré-incrémentation `++x`, `y` aurait reçu `6` directement.

Question 11. Réponse : **B)**

2

En C++, la division entre deux entiers est une division entière (euclidienne). Le résultat de `5 / 2` est donc `2`, la partie décimale étant tronquée. Pour obtenir `2.5`, il faudrait qu'au moins un des opérandes soit un flottant, par exemple `5 / 2.0f` ou `float(5) / 2`.

Question 12. Réponse : **B)**

Il déclare une variable dont la valeur ne peut plus être modifiée après initialisation

Le mot-clé `const` indique que la variable est constante : sa valeur doit être initialisée lors de la déclaration et ne peut plus être modifiée par la suite. Toute tentative de modification provoquera une erreur de compilation. Cela améliore la sûreté du code et peut permettre au compilateur d'effectuer des optimisations.

Question 13. Réponse : **B)**

3

En C++, les arguments sont passés par copie par défaut. La fonction `increment` reçoit une copie de `x`, donc la modification `a = a + 1` n'affecte que la copie locale. La variable `x` dans `main` reste inchangée à la valeur 3. Pour modifier `x`, il faudrait passer par référence avec `void increment(int& a)`.

Question 14. Réponse : **C)**

`void modifier(int& a)`

La signature `void modifier(int& a)` utilise une référence, ce qui permet à la fonction de modifier directement la variable originale. Le passage par copie (option A) ne modifie pas l'original. L'option B utilise un pointeur, ce qui fonctionne aussi mais n'est pas la manière idiomatique en C++ moderne. L'option D utilise une référence constante qui interdit la modification.

Question 15. Réponse : **B)**

4

Le vecteur est initialisé avec 3 éléments `{10, 20, 30}`, puis `push_back(40)` ajoute un quatrième élément à la fin. La méthode `size()` retourne donc 4. Le `std::vector` est un tableau dynamique qui peut changer de taille à l'exécution, contrairement à `std::array`.

Question 16. Réponse : **C)**

Les membres d'une `struct` sont publics par défaut, ceux d'une `class` sont privés par défaut

La seule différence technique entre `struct` et `class` en C++ est la visibilité par défaut des membres. Dans une `struct`, les membres sont publics par défaut, tandis que dans une `class`, ils sont privés par défaut. Les deux peuvent contenir des méthodes, des attributs, des constructeurs et utiliser l'héritage. Par convention, on utilise `struct` pour des objets simples et `class` pour des objets avec encapsulation.

Question 17. Réponse : **D)**

Erreur de compilation (i n'est plus accessible)

La variable `i` est déclarée dans le bloc d'initialisation de la boucle `for`, ce qui signifie que sa durée de vie (scope) est limitée au bloc de la boucle. Après l'accolade fermante de la boucle, `i` n'existe plus. Tenter de l'utiliser après provoque une erreur de compilation. Ce comportement diffère de Python où la variable resterait accessible.

Question 18. Réponse : **B)**

`for (int v : valeurs)`

La boucle for étendue (range-based for), introduite en C++11, utilise la syntaxe `for (type variable : conteneur)`. Le caractère `:` sépare la variable d'itération du conteneur. Les syntaxes `in`, `of` et `for each` n'existent pas en C++ standard. Cette syntaxe est pratique pour parcourir tous les éléments d'un conteneur sans gérer d'indice.

Question 19. Réponse : **C)**

`int`, `float`, `double`

Avec `auto`, le compilateur déduit le type à partir de la valeur d'initialisation. `5` est un littéral entier, donc `a` est `int`. `8.4f` a le suffixe `f` qui indique un `float`, donc `b` est `float`. `4.2` sans suffixe est par défaut un `double` en C++, donc `c` est `double`. La distinction entre `float` et `double` dépend du suffixe `f`.

Question 20. Réponse : **A)**

`g++ -o prog main.cpp`

La commande `g++ -o prog main.cpp` lance le compilateur C++ `g++`, compile le fichier source `main.cpp` et produit un exécutable nommé `prog` grâce à l'option `-o`. L'option B utilise `gcc` qui est le compilateur C (et non C++). Les options C et D ne correspondent pas à des syntaxes valides de g++.

Question 21. Réponse : **B)**

`1 2 3`

Le vecteur `v` est passé par copie à la fonction `multiply`. La multiplication est donc effectuée sur une copie locale du vecteur, et le vecteur original dans `main` reste inchangé. L'affichage produit donc `1 2 3`. Pour que la modification soit effective, il faudrait passer le vecteur par référence : `void multiply(std::vector<float>& vec, float s)`.

Question 22. Réponse : **B)**

`20 10`

Le C++ autorise la déclaration de variables portant le même nom dans des blocs différents (shadowing). Dans le bloc interne, `x` vaut 20, donc le premier affichage est `20`. À la sortie du bloc interne, la variable `x` du bloc de `main` (valeur 10) redevient visible, d'où le second affichage `10`. La variable globale `x = 5` est masquée par les variables locales et n'est jamais affichée.

Question 23. Réponse : **B)**

La fonction `addition` est utilisée avant d'être déclarée

En C++, une fonction doit être déclarée (au minimum sa signature) avant d'être utilisée. Ici, `addition` est appelée dans `main` avant d'être définie, ce qui provoque une erreur de compilation. Pour corriger, on peut soit déplacer la définition de `addition` avant `main`, soit ajouter une déclaration anticipée `int addition(int a, int b);` avant `main`.

Question 24. Réponse : **A)**

`5 5`

Il n'y a pas de conflit de noms car `v.norm()` appelle la méthode membre de la structure `vec3`, tandis que `norm(v)` appelle la fonction libre qui prend une référence constante vers `vec3`. Les deux calculent la norme euclidienne de `(3, 4, 0)`, soit `sqrt(9 + 16 + 0) = sqrt(25) = 5`. Le C++ distingue bien les fonctions membres des fonctions non-membres.

Question 25. Réponse : **B)**

-3

L'appel `solve(2, 6)` passe deux arguments entiers, ce qui correspond à la première surcharge `int solve(int a, int b)`. Le calcul effectué est `-6 / 2 = -3`, et le type de retour est `int`. Avec `auto`, la variable `x` est donc de type `int` et vaut `-3`. L'affichage est `-3` (sans point décimal car c'est un entier). La seconde surcharge n'est pas appelée car elle attend trois arguments de type `float`.