Lambda Expressions

Question 1 : Qu'est-ce qu'une fonction lambda en C++ ?

Une fonction anonyme définie à l'endroit où elle est invoquée

```
Une fonction anonyme définie à l'endroit où elle est appelée Explication : Les
fonctions Lambda en C++ sont des fonctions anonymes qui peuvent être définies à
l'endroit où elles sont appelées ou invoquées.
```

Question 2 : Qu'est-ce qui est capturé dans une expression lambda qui utilise la clause de capture `[&]` ?

Toutes les variables locales par référence

```
La clause de capture [&] dans une expression lambda indique au compilateur de
capturer toutes les variables locales disponibles dans la portée par référence.
```

Question 3 : Qu'imprimera le code suivant ?

```
int a = 5;
auto lambda = [a]() mutable { a = 10; return a; };
cout << a << " " << lambda() << endl;
```

5 10

```
La fonction lambda capture « a » par valeur. Ainsi, le changement en « a » à
l'intérieur de la fonction lambda n'affectera pas le « a » à l'extérieur. Ainsi,
le « a » d'origine reste « 5 » et la fonction lambda renvoie « 10 ».
```

Question 4 : Lequel des éléments suivants `n'est PAS` un composant obligatoire d'une expression lambda en `C++` ?

Déclaration du type de retour

```
La déclaration du type de retour n'est pas nécessaire dans une fonction lambda en
C++. Le compilateur peut déduire automatiquement le type de retour. Cependant, si
besoin, il peut être fourni après la liste des paramètres avec l'opérateur '->'.
```

Question 5 : Qu'indique la clause de capture `[=]` dans une expression lambda ?

Capturez toutes les valeurs locales par défaut

```
La clause de capture [=] dans une expression lambda indique au compilateur de
capturer toutes les variables locales disponibles dans la portée par valeur.
```

Question 6 : Que fait le mot-clé `mutable` dans une expression lambda ?

Permet de modifier les variables capturées

```
En C++, les fonctions lambda qui capturent les variables par valeur les traitent
comme des const dans le corps du lambda. Le mot-clé mutable permet de modifier ces
variables capturées dans le lambda.
```

Question 7 : Quel est le type de retour d'une expression lambda si aucun type de retour n'est spécifié ?

Il est déduit par le compilateur sur la base de l'instruction return

```
Si aucun type de retour n'est spécifié pour une expression lambda, le compilateur
déduira le type de retour en fonction de l'instruction return dans le corps de la
fonction.
```

Question 8 : Lequel des éléments suivants provoquera une erreur de compilation ?

```
A) auto lambda = [](int a, int b){ return a + b; };

B) auto lambda = [](){ return "Hello, World!"; };

C) auto lambda = []=(){ return 5; };

D) auto lambda = [&](int a, int b){ return a + b; };
```

C

```
Il n'y a pas de clause de capture comme []=. Les clauses de capture correctes sont
[] (ne capture rien), [&] (capture tout par référence) et [=] (capture tout par
valeur).
```

Question 9 : Quel opérateur est utilisé pour appeler une fonction lambda ?

()

```
L'opérateur () est utilisé pour appeler ou invoquer une fonction lambda, un peu
comme une fonction normale.
```

Question 10 : Dans lequel des algorithmes `STL` suivants pourriez-vous utiliser une fonction lambda ?

Tout ce qui précède

```
Vous pouvez utiliser des expressions lambda comme prédicats ou fonctions
personnalisées pour les algorithmes STL comme std::for_each, std::sort et
std::transform.
```

Question 11 : Étant donné le code suivant, quel est le résultat ?

```
vector<int> vec  {2, 4, 6, 8, 10};

auto it = find_if(vec.begin(), vec.end(), [](int num){ return num > 5; });

cout << *it << endl;
```

6

```
La fonction lambda utilisée dans std::find_if recherche le premier nombre du
vecteur supérieur à 5. Elle trouve « 6 », qui est le premier nombre répondant à
cette condition.
```

Question 12 : Comment utiliser une fonction lambda pour trier un vecteur d'entiers par ordre décroissant à l'aide de la fonction `std :: sort` ?

```
std::sort(vec.begin(), vec.end(), [](int a, int b){return a > b;});
```

```
La fonction lambda fournie comme troisième argument de la fonction std::sort sert
de comparateur personnalisé. Il doit renvoyer « vrai » si le premier argument doit
être classé avant le second. Dans ce cas, pour trier par ordre décroissant, nous
devons renvoyer « vrai » lorsque « a » est supérieur à « b ».
```

Question 13 : Quelle fonction STL pouvez-vous utiliser avec une fonction lambda pour appliquer une opération à chaque élément d'un conteneur ?

```
std::transform
```

```
La fonction std::transform applique une fonction donnée à une plage et stocke le
résultat dans une autre plage, qui peut être la même que la première plage.
```

Question 14 : Étant donné le code suivant, quel est le résultat ?

```
vector<int> vec  {1, 2, 3, 4, 5};

int sum = std::accumulate(vec.begin(), vec.end(), 0, [](int total, int x){ return
total + x; });

cout << sum << endl;
```

15

```
std::accumulate est utilisé ici avec une fonction lambda qui additionne tous les
éléments du vecteur. Le total initial est « 0 » et la fonction lambda ajoute
chaque élément de « vec » à ce total. La somme des nombres de 1 à 5 est 15.
```

Question 15 : Quel est le résultat du code suivant :

```
#include <iostream>
using namespace std;

class MyClass {
public:
    int val = 10;

    void func() {
        int local_val = 20;
        auto lambda = [this, local_val]() {
            return val * local_val;
        };
        cout << lambda() << endl;
    }
};

int main() {
    MyClass obj;
    obj.func();
    return 0;
}
```

200

Le code capture le pointeur 'this' de l'instance de classe MyClass et une variable locale local_val par valeur dans la fonction lambda à l'intérieur de func(). La fonction lambda multiplie le membre val de MyClass (qui vaut 10) par local_val (qui vaut 20), ce qui donne 200.