24 Essential C++ Interview Questions *

toptal.com/c-plus-plus/interview-questions

What will the line of code below print out and why?

```
#include <iostream>

int main(int argc, char **argv)
{
    std::cout << 25u - 50;
    return 0;
}
```

The answer is *not* -25. Rather, the answer (which will surprise many) is 4294967271, assuming 32 bit integers. Why?

In C++, if the types of two operands differ from one another, then the operand with the “lower type” will be promoted to the type of the “higher type” operand, using the following type hierarchy (listed here from highest type to lowest type): long double, double, float, unsigned long int, long int, unsigned int, int (lowest).

So when the two operands are, as in our example, `25u` (unsigned int) and `50` (int), the `50` is promoted to also being an unsigned integer (i.e., `50u`).

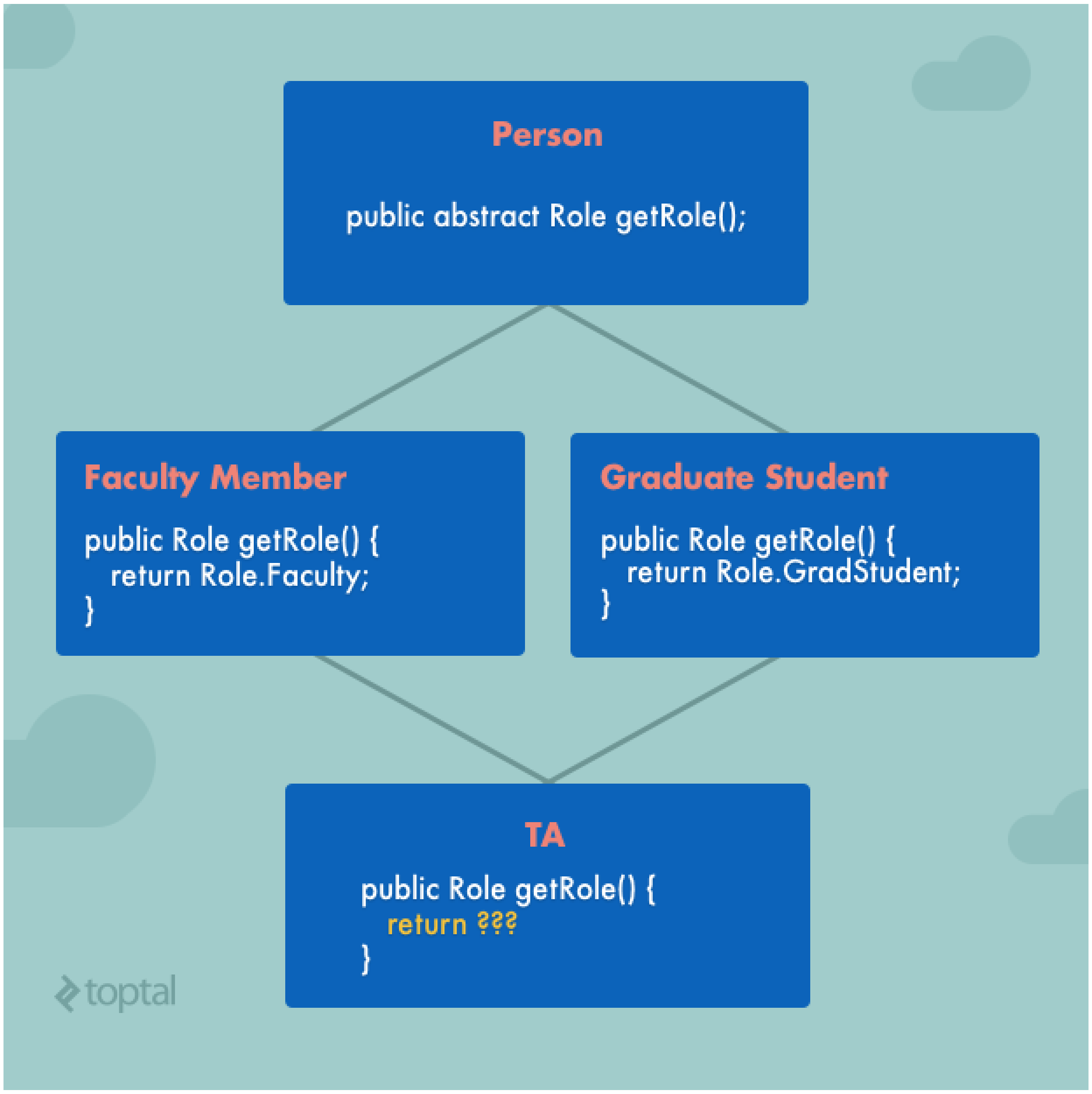
Moreover, the result of the operation will be of the type of the operands. Therefore, the result of `25u - 50u` will itself be an unsigned integer as well. So the result of `-25` converts to 4294967271 when promoted to being an unsigned integer.

C++ supports multiple inheritance. What is the “diamond problem” that can occur with multiple inheritance? Give an example.

It means that we cannot create hybrid inheritance using multiple and hierarchical inheritance.

Let’s consider a simple example. A university has people who are affiliated with it. Some are students, some are faculty members, some are administrators, and so on. So a simple inheritance scheme might have different types of people in different roles, all of whom inherit from one common “Person” class. The Person class could define an abstract `getRole()` method which would then be overridden by its subclasses to return the correct role type.

But now what happens if we want to model the role of a Teaching Assistant (TA)? Typically, a TA is *both* a grad student *and* a faculty member. This yields the classic diamond problem of multiple inheritance and the resulting ambiguity regarding the TA's `getRole()` method:



(Note the diamond shape of the above inheritance diagram, hence the name.)

Which `getRole()` implementation should the TA inherit? That of the Faculty Member or that of the Grad Student? The simple answer might be to have the TA class override the `getRole()` method and return newly-defined role called "TA". But that answer is also imperfect as it would hide the fact that a TA is, in fact, both a faculty member and a grad student.

What is the error in the code below and how should it be corrected?

```
my_struct_t *bar;

memset(bar, 0, sizeof(bar));
```

The last argument to `memset` should be `sizeof(*bar)`, not `sizeof(bar)`. `sizeof(bar)` calculates the size of `bar` (i.e., the pointer *itself*) rather than the size of the structure pointed to by `bar`.

The code can therefore be corrected by using `sizeof(*bar)` as the last argument in the call to `memset`.

A sharp candidate might point out that using `*bar` will cause a dereferencing error if `bar` has not been assigned. Therefore an even safer solution would be to use `sizeof(my_struct_t)`. However, an even sharper candidate must know that in this case using `*bar` is absolutely safe within the call to `sizeof`, even if `bar` has not been initialized yet, since `sizeof` is a compile time construct.

Find top C++ developers today. Toptal can match you with the best engineers to finish your project.

Hire Toptal's C++ developers
What will `i` and `j` equal after the code below is executed? Explain your answer.

```
int i = 5;
int j = i++;
```

After the above code executes, `i` will equal 6, but `j` will equal 5.

Understanding the reason for this is fundamental to understanding how the unary increment (`++`) and decrement (`--`) operators work in C++.

When these operators *precede* a variable, the value of the variable is modified first and *then* the modified value is used. For example, if we modified the above code snippet to instead say `int j = ++i;`, `i` would be incremented to 6 and *then* `j` would be set to that modified value, so both would end up being equal to 6.

However, when these operators *follow* a variable, the unmodified value of the variable is used and *then* it is incremented or decremented. That's why, in the statement `int j = i++;` in the above code snippet, `j` is first set to the unmodified value of `i` (i.e., 5) and *then* `i` is incremented to 6.

Assuming `buf` is a valid pointer, what is the problem in the code below? What would be an alternate way of implementing this that would avoid the problem?

```
size_t sz = buf->size();
while ( --sz >= 0 )
{

}
```

The problem in the above code is that `--sz >= 0` will *always* be true so you'll never exit the `while` loop (so you'll probably end up corrupting memory or causing some sort of memory violation or having some other program failure, depending on what you're doing inside the loop).

The reasons that `--sz >= 0` will *always* be true is that the type of `sz` is `size_t`. size_t is really just an alias to one of the fundamental unsigned integer types. Therefore, since `sz` is unsigned, it can *never* be less than zero (so the condition can never be true).

One example of an alternative implementation that would avoid this problem would be to instead use a `for` loop as follows:

```
for (size_t i = 0; i < sz; i++)
{

}
```

Consider the two code snippets below for printing a vector. Is there any advantage of one vs. the other? Explain.

Option 1:

```
vector vec;

for (auto itr = vec.begin(); itr != vec.end(); itr++) {
        itr->print();
}
```

Option 2:

```
vector vec;

for (auto itr = vec.begin(); itr != vec.end(); ++itr) {
        itr->print();
}
```

Although both options will accomplish precisely the same thing, the second option is better from a performance standpoint. This is because the post-increment operator (i.e., `itr++`) is more expensive than pre-increment operator (i.e., `++itr`). The underlying implementation of the post-increment operator makes a copy of the element before incrementing it and then returns the copy.

That said, many compilers will automatically optimize the first option by converting it into the second.

Implement a template function `IsDerivedFrom()` that takes class C and class P as template parameters. It should return true when class C is derived from class P and false otherwise.

Implement a template boolean `IsSameClass()` that takes class A and B as template parameters. It should compare class A and B and return false when they are different classes and true if they are the same class.

Is it possible to have a recursive inline function?

What is the output of the following code:

```
#include <iostream>

class A {
public:
    A() {}
    ~A() {
        throw 42;
    }
};

int main(int argc, const char * argv[]) {
    try {
        A a;
        throw 32;
    } catch(int a) {
        std::cout << a;
    }
}
```

You are given library class Something as follows:

```
class Something {
public:
    Something() {
        topSecretValue = 42;
    }
    bool somePublicBool;
    int somePublicInt;
    std::string somePublicString;
private:
    int topSecretValue;
};
```

Implement a method to get topSecretValue for any given Something* object. The method should be cross-platform compatible and not depend on sizeof (int, bool, string).

Implement a void function F that takes pointers to two arrays of integers (`A` and `B`) and a size `N` as parameters. It then populates `B` where `B[i]` is the product of all `A[j]` where `j != i` .

For example: If `A = {2, 1, 5, 9}` , then `B` would be `{45, 90, 18, 10}` .

When you should use virtual inheritance?

Is there a difference between **class** and **struct**?

What is the output of the following code:

```
#include <iostream>

int main(int argc, const char * argv[]) {
    int a[] = {1, 2, 3, 4, 5, 6};
    std::cout << (1 + 3)[a] - a[0] + (a + 1)[2];
}
```

What is the output of the following code:

```
#include <iostream>

class Base {
    virtual void method() {std::cout << "from Base" << std::endl;}
public:
    virtual ~Base() {method();}
    void baseMethod() {method();}
};

class A : public Base {
    void method() {std::cout << "from A" << std::endl;}
public:
    ~A() {method();}
};

int main(void) {
    Base* base = new A;
    base->baseMethod();
    delete base;
    return 0;
}
```

Explain the `volatile` and `mutable` keywords.

How many times will this loop execute? Explain your answer.

```
unsigned char half_limit = 150;

for (unsigned char i = 0; i < 2 * half_limit; ++i)
{
    // do something;
}
```

How can you make sure a C++ function can be called as e.g. `void foo(int, int)` but not as any other type like `void foo(long, long)` ?

What is the problem with the following code?

```
class A
{
public:
A() {}
~A(){}
};

class B: public A
{
public:
B():A(){}
~B(){}
};

int main(void)
{
  A* a = new B();
  delete a;
}
```

Are you allowed to have a `static const` member function? Explain your answer.

What is a storage class?

How can a C function be called in a C++ program?

What will be the output of the following program?

```
#include <iostream>

struct A
{
    int data[2];

    A(int x, int y) : data{x, y} {}
    virtual void f() {}
};

int main(int argc, char **argv)
{
    A a(22, 33);

    int *arr = (int *) &a;
    std::cout << arr[2] << std::endl;

    return 0;
}
```

* There is more to interviewing than tricky technical questions, so these are intended merely as a guide. Not every “A” candidate worth hiring will be able to answer them all, nor

does answering them all guarantee an “A” candidate. At the end of the day, hiring remains an art, a science — and a lot of work.
Submit an interview question

Submitted questions and answers are subject to review and editing, and may or may not be selected for posting, at the sole discretion of Toptal, LLC.

Looking for C++ experts? Check out Toptal’s C++ developers.
View full profile »

Mário José Bittencourt
Brazil

Mário is a software engineer and architect with more than 30 years of providing software support to several companies in the market. He specializes in C/C++ development. He considers himself a back-end developer and is currently focused on Windows and MicroChip PIC platforms. Mário is a professional who has worked remotely and is an excellent communicator.

Hire Mário
View full profile »

Andrew Schuessler
United States

Andrew is an experienced and talented C/C++ engineer. His experience ranges from low-level device driver development and team leadership delivering of software systems from scratch for state-of-the-art fitness electronics to development and production deployment of multi-threaded/multi-core HF trading applications. He loves to build complex, robust, performant and scalable infrastructure as well as impactful end-user applications.

Hire Andrew
View full profile »

Mohit Jain
India

Mohit is currently focusing on his full time job and not looking forward for freelancing work.

Hire Mohit