10 Proven C++ Interview Questions to Ask

tests4geeks.com/blog/cpp-interview-questions

May 25, 2017

Need to hire C++ developers? Not confident you have the resources to pick the best ones? **Don't worry – we're here to help you**!

We've assembled a team of expert C++ programmers who have worked hard to produce a collection of premade C++ interview questions you can use to bolster your interview process.

These **C++ programming questions** are your secret weapon for face-to-face interviews with prospective hires: **answers, explanations, follow-up questions**, code snippets. Even if you're not a C++ guru yourself, these questions will let you conduct an interview like one, helping you find the master developers your project demands!

So here are the questions and answers:

Recommendation

If your company is hiring a C++ programmer, check out this C++ test prepared by our professionals. It will allow you to identify the **best talents** very easily!

[Question #1 – Smart Pointer – Standard library]

What happens when a **std::unique_ptr** is passed by value to a function? For example, in this code snippet?

```
#include <memory>

auto f(std::unique_ptr<int> ptr) {
  *ptr = 42;
  return ptr;
}

int main() {
  auto ptr = std::make_unique<int>();
  ptr = f(ptr);
}
```

Why this C++ question?

As mentioned before, **memory management** is a nontrivial burden for the C++ programmer. Smart pointers are helpful in this regard, but they must be well understood in order to be used correctly. This question tests for the interview candidate's understanding of one common type of **smart pointer**.

Possible answers

The correct answer is that this code won't even compile. The **std::unique_ptr** type cannot be copied, so passing it as a parameter to a function will fail to compile.

To convince the compiler that this is fine, **std::move** can be used:

```
ptr = f(std::move(ptr));
```

Follow-up questions

The interview candidate might think that returning a **noncopiable** object from a function is also a compiler error, but in this case it's allowed, thanks to copy elision. You can ask the candidate under what conditions copy elision is performed.

Of course, the above construct with **std::move** is less than ideal. Ask the candidate how they would change the function **f** to make it better. For example, passing a (const) reference to the **unique_ptr**, or simply a reference to the **int** pointed to, is probably preferred.

↑↑ Scroll up to the list of C++ questions

[Question #2 – Rule of Five – RAII]

Write a **copy constructor, move constructor, copy assignment operator**, and **move assignment operator** for the following class (assume all required headers are already included):

```
class DirectorySearchResult {
public:
  DirectorySearchResult(
    std::vector<std::string> const& files,
    size_t attributes,
    SearchQuery const* query)
    : files(files),
      attributes(attributes),
      query(new SearchQuery(*query))
  { }

  ~DirectorySearchResult() { delete query; }

private:
  std::vector<std::string> files;
  size_t attributes;
  SearchQuery* query;
};
```

Why this interview question?

Writing boilerplate like this should be straightforward for any **C++ programmer** (treat this question as one of the c++ interview questions for freshers).

It is also interesting to see the interview candidate's response to the class design, and see if they question it at all.

Possible answers

Copy constructor:

```
DirectorySearchResult(DirectorySearchResult const& other)
  : files(other.files),
    attributes(other.attributes),
    query(other.query ? new SearchQuery(*other.query) : nullptr)
{ }
```

Here, it's the check for null pointer to watch out for. As given, the **query** field cannot be null, but since it's not **const** this may change later.

Move constructor:

```
DirectorySearchResult(DirectorySearchResult&& other)
  : files(std::move(other.files)),
    attributes(other.attributes),
    query(other.query)
{
  other.query = nullptr;
}
```

Watch out for correct usage of **std::move** here, as well as correct “pointer stealing” for the **query** pointer. It must be nulled, otherwise the object will be deleted by **other**.

Assignment operator:

```
DirectorySearchResult& operator=(DirectorySearchResult const& other)
{
  if (this == &other)
    return *this;
  files = other.files;
  attributes = other.attributes;
  delete query;
  query = other.query ? new SearchQuery(*other.query) : nullptr;
  return *this;
}
```

A pitfall is forgetting to check for self-assignment. It’s also worth looking out for a correct function signature, and again handling a null **query**.

Move assignment operator:

```
DirectorySearchResult& operator=(DirectorySearchResult&& other)
{
  files = std::move(other.files);
  attributes = other.attributes;
  std::swap(query, other.query);
  return *this;
}
```

As with the move constructor, watch out for correct **std::move** usage and correct pointer stealing.

Follow-up questions

If the interview candidate hasn’t mentioned it already, ask them how the design of this class could be improved. There is no reason for **SearchQuery** to be a pointer! If we make it a simple object (composition), the default, compiler-generated versions of all four functions would suffice, and the destructor can be removed as well.

10 Proven C++ Programming Questions to Ask on Interview (Explanations, Possible

Answers, Following Questions)Click To Tweet
↑↑ Scroll up to the list of C++ questions

[Question #3 – Finding bugs – Basics]

There are multiple issues/bugs with the following code. Name as many as you can!

```
#include <vector.h>

void main(int argc, char** argv)
{
  int n;
  if (argc > 1)
    n = argv[0];
  int* stuff = new int[n];
  vector<int> v(100000);
  delete stuff;
  return 0;
}
```

Why this programming question?

In any programming language, debugging is an essential skill; C++ is no exception. Being able to debug a program on paper, without looking at its actual runtime behavior, is a useful skill, because the ability to spot incorrect code helps the programmer avoid those mistakes in their own code. Also, it's just plain fun to pick someone else's code apart like that, so this serves as a good warm-up question to put interview candidates at ease.

Possible answers

- **vector.h** should be **vector**
- **main** cannot be **void** in C++
- **argv[0]** is the program name, not the first argument
- **argv[0]** is a pointer to a string, and should not be assigned to **n** directly
- If **argc <= 1**, then **n** is uninitialized, and using it invokes undefined behavior
- **vector** is used without **using namespace std** or **std::**
- the **vector** constructor might throw an exception (**std::bad_alloc**), causing **stuff** to be leaked
- **stuff** points to an array, so it should be deleted using **delete[]**
- cannot return **0** from a **void** function

Follow-up questions

For each issue the candidate identifies, ask how it can best be fixed. They should at least mention using a smart pointer or **std::vector** instead of a raw pointer for **stuff**.

↑↑ Scroll up to the list of C++ questions

[Question #4 – Automatic objects – RAII]

Explain what an **automatic object** is (that is, an object with automatic storage duration; also called **"Stack object"**) and what its lifetime is.

Explain how an object with dynamic storage duration (**heap object**) is created, and how it is destroyed. Why is dynamic storage duration discouraged unless necessary, and where is it necessary?

What is the inherent problem with raw pointers owning an object? I.e. why is the following considered bad practice, and what standard library construct would you utilize if you needed a dynamically resizable array?

```
auto p = new int[50];
```

Show how to initialize a **smart pointer**, and explain why using one is exception safe.

Why this C++ interview question?

Unlike garbage-collected languages, C++ puts the burden of managing object lifetimes (and thereby memory) on the programmer. There are many ways to do this wrong, some ways to do it approximately right, and few ways to do it entirely "by the book". This series of questions drills the interview candidate about these matters.

Possible answers

A **stack object** is created at the point of its definition, and lives until the end of its scope (basically, until the closing curly brace of the block it is declared in). A **heap object** is created with the **new** operator and lives until **delete** is called on it.

The problem with raw pointers is that ownership is not enforced; it is the responsibility of the programmer to ensure that the object pointed to is deleted, and deleted only once. Advanced candidates might also mention exception safety here, because the possibility of exceptions makes it significantly more complicated to ensure eventual deletion.

Unlike its precursor C, C++ offers **smart pointers**, which are the preferred tool for the job. In particular, to create a "smart pointer" to a dynamically resizable array, **std::vector** should be used.

An example of smart pointer usage:

```
auto p = std::make_unique<Foo>();
```

Follow-up questions

Ask the candidate which types of smart pointer exist in the C++ standard library, and what their differences are. Ask which ones can be used in standard containers (e.g. **vector, map**).

You can also ask about the difference between **std::make_shared<T>(...)** and **std::shared_ptr<T>(new T(...))**. (The former is more exception-safe when used as a function argument, and might also be implemented more efficiently.)

↑↑ Scroll up to the list of C++ questions

[Question #5 – Iterators – Standard library]

The C++ standard library represents ranges using **iterators**. What is an iterator, and what different kinds do you know of?

Can you explain why the following snippet fails, and why **l's** iterators aren't suitable?

```
std::list<int> l {1, 2, 3};
std::sort(l.begin(), l.end());
```

Explain how the begin and end iterators of a range correspond to its elements and illustrate this by giving the expressions for begin and end iterators of an array **arr**.

Why this question?

Standard library containers are the bread and butter of writing algorithms in C++. As in any programming language, one of the most common tasks to perform on a container is to iterate over it. In the C++ standard library, this is accomplished using special-purpose, pointer-like objects called iterators, which come in different types. Asking the candidate about these will reveal how well they understand the concept of iterators, as well as the structure of the underlying container.

Possible answers

An iterator resembles a smart pointer, in the sense that it points to a particular object in a container. But iterators have additional operations besides deferencing, depending on their type: forward iterators can be incremented, bidirectional iterators can additionally be decremented, and random access iterators can additionally be moved by an arbitrary offset. There are also output iterators, which may for example add objects to the container when assigned to.

The reason that the **sort** call won't work is that it requires a random access iterator, but **std::list** only provides a bidirectional iterator.

By convention, the **begin** iterator of a collection refers to the first element, and the **end** iterator refers one past the last element. In other words, they form a half-open range: **[begin, end)**.

Follow-up questions

Ask how the code could be fixed to sort an **std::list** (e.g. by copying it into a vector first, and back again after sorting). You could even ask the candidate to implement an iterator for a particular data structure (e.g. an array).

10 Proven C++ Programming Questions to Ask on Interview (Explanations, Possible Answers, Following Questions)Click To Tweet
↑↑ Scroll up to the list of C++ questions

[Question #6 – Undefined/unspecified behavior – Standards]

Describe what **"undefined behavior"** means, and how it differs from **"unspecified behavior"**. Give at least 3 examples of undefined behavior.

Why this C++ question?

The C++ standard does not specify the behavior of the program in every case, and deliberately leaves some things up to compiler vendors. Typically, such cases are to be avoided in practice, so this question is to test whether the interview candidate has seen practical examples of such code.

Possible answers

Undefined behavior (UB) means that the standard guarantees nothing about how the program should behave. Unspecified (or implementation-defined) behavior means that the standard requires the behavior to be well-defined, but leaves the definition up to the compiler implementation.

This is only the textbook definition; candidates should mention that undefined behavior implies that anything might happen: the program works as intended, it crashes, it causes demons to fly out of your nose. They should mention that UB should always be avoided. They might mention that implementation-defined behavior should probably be avoided as well.

Common examples of undefined behavior include:

- dereferencing a null or wild pointer
- accessing uninitialized memory, like going beyond the bounds of an array or reading an uninitialized local variable

- deleting the same memory twice, or more generally deleting a wild pointer
- arithmetic errors, like division by zero

Follow-up questions

If the candidate doesn't come up with enough UB cases, you can make up some cases of dodgy-looking code and ask them whether it exhibits UB or not.

↑↑ Scroll up to the list of C++ questions

[Question #7 – Macros – Preprocessor]

At what stage of compilation is the **preprocessor invoked**, and what kind of directives are there?

The following is the declaration of a macro that is used as a constant in some internal API header (**B** is another entity):

```
#define A 2048*B
```

List two issues with this macro: one related to this particular one, for which you should give illustrative example code that breaks the macro, and one related to all macros (hint: think of names).

Why this question?

Even though the preprocessor is typically used in C++ for just a few specific tasks, it is still important to have a basic understanding of its operation and its limitations. The preprocessor makes it very easy to shoot yourself in the foot, so "responsible usage" is essential.

As stated, this looks like more of a trivia than a discussion question, and it's up to the interviewer to dig deeper where necessary.

Possible answers

The preprocessor is invoked on a translation unit ("source file") before actual compilation starts. The output of the preprocessor is passed to the compiler. Even junior candidates should give an answer along these lines.

Common preprocessor directives are **#include, #define, #ifdef, #ifndef, #if, #else, #elif, #endif**. Candidates should be able to list most of these. They might also mention less common directives, such as **#undef** and **#pragma**.

The two problems with the **#define** code are:

- Lack of parentheses, in two places. If **B** is defined as **1+1**, then **A** will not have the value **4096** as expected, but rather **2049**. If **A** is used in an expression like **!A**, this will expand to **~2048*B**, rather than **~(2048*B)**, which may have a very different value.
  The macro should have been defined as:

  ```
  #define A (2048*(B))
  ```

  Good candidates will mention that this should probably not have been a macro in the first place, but simply a compile-time constant.

- Overly short names. Preprocessor macros are all in a single scope, which spans all files **#included** afterwards as well, so one has to be very careful about name clashes. If some unrelated code declared an **enum { A, B }**, for example, that code would fail to compile with a very confusing error message.

Follow-up questions

It is common for candidates to mention only one of the two pairs of missing parentheses. In this case, prompt them to find more issues. This can also lead to a discussion about why the preprocessor should be avoided when possible, and what the C++ style alternatives are.

↑↑ Scroll up to the list of C++ questions

[Question #8 – Pointer detector – Templates]

Write a templated struct that determines, at compile time, whether its template argument **T** is a pointer.

Why this C++ programming question?

Template metaprogramming in C++ is an advanced topic, so this question is one of the c++ interview questions for experienced professionals and should not be posed to junior interview candidates. However, for senior candidates, this question can be a good indicator of the depth of their practical experience with the C++ language.

Possible answers

The candidate might mention that **std::is_pointer** already exists. It could be implemented like this:

```
template<typename T>
struct is_pointer {
  enum { value = false; };
};
template<typename T>
struct is_pointer<T*> {
  enum { value = true; };
}
```

Template overload resolution will pick the most specific version, so if the type is a pointer, the last one will be selected, which contains an enum field **value** with the value **true**. Otherwise it falls back to the first, where **value** is **false**.

It is also possible to use a **static const bool** instead of an **enum**, but this has some drawbacks. The constants would still occupy memory space, so it's not a 100% compile-time construct anymore. Moreover, you'd need to redefine the existence of **value** outside the template in order for it to exist, because the assignment of a value in this case does not make it into an actual definition. It would work in some cases, but would fail if you take the address, for example.

Follow-up questions

If the candidate doesn't offer an explanation of their code, ask them for it. You can also ask them about what "most specific" means, i.e. how template overload resolution actually works.

Please note one more time that this question is one of the advanced c++ interview questions.

10 Proven C++ Programming Questions to Ask on Interview (Explanations, Possible Answers, Following Questions)Click To Tweet
↑↑ Scroll up to the list of C++ questions

[Question #9 – Insertion sort – Templates]

Define a function **insertion_sort** which accepts as first and only argument a reference to an **std::array** only if the element types are integral (the trait **std::is_integral** might be of help) and the size of the array is less than 128 elements, and sorts it using insertion sort.

Why this C++ technical interview question?

This tests for the candidate's knowledge of **std::enable_if**, a compile-time construct which lets the C++ programmer put additional restrictions on the types that their template accepts. This is an advanced skill, useful when writing library code, for example to avoid incorrect or inefficient usage of the API.

The interesting part here is the function signature, but the candidate's ability to implement an insertion sort is also tested. It's up to the interviewer how much emphasis to put on either of these parts.

Possible answers

A possible implementation:

```
template<typename T, std::size_t N>
typename std::enable_if<
    N < 128 && std::is_integral<T>::value,
    void
>::type insertion_sort(std::array<T, N>& array) {
  for (std::size_t i = 0; i < N; i++) {
    for (std::size_t j = i; j > 0 && array[j] < array[j-1]; j--) {
      std::swap(array[j], array[j - 1]);
    }
  }
}
```

Do not punish the candidate for not knowing the exact usage of all these standard library templates. The important thing is that they grasp the overall concepts; the details can be looked up online easily enough.

Follow-up questions

If you haven't asked the "**pointer detector**" question, you could ask the candidate here how they would implement **std::enable_if** and/or **std::is_integral**.

↑↑ Scroll up to the list of C++ questions

[Question #10 – Max heap – Algorithms and data structures]

Describe what a **max heap** is. Provide the definition of a max heap class which supports a wide range of element types and all basic operations that can be performed on a heap.

Why this question?

At first, this seems like a pure algorithms question, but note that we are not asking for the **implementation** of any of the operations. This question purely tests the candidate's ability to design a proper C++ class.

Possible answers

Depending on the design decisions that the interview candidate makes along the way, the result could be something like this:

```
template<typename T>
class heap {
  public:
    void add(T const &value);
    T const &max() const;
    T remove_max();
    size_t size() const;
  private:
    std::vector<T> elements;
};
```

Look out for the following:

- Are the class and its operations being named consistently and intuitively?
- What type is being used for the internal container? **std::vector** is preferred, but other possibilities exist. Use of a raw pointer to an array is a red flag, because it will make the class needlessly hard to implement.
- What is the argument type of the **add** function? This should be a pointer or reference type in order to avoid needless copying. An overload that takes an **rvalue** reference is a bonus.
- What is the return value of the **max** and **remove_max** functions?
- Are functions marked as **const** where possible?
- Are **noexcept** clauses used as appropriate?

Follow-up questions

Many design decisions can be made along the way, each of which can be used by the interviewer as a hook to lead into further discussion about the various tradeoffs in the design.

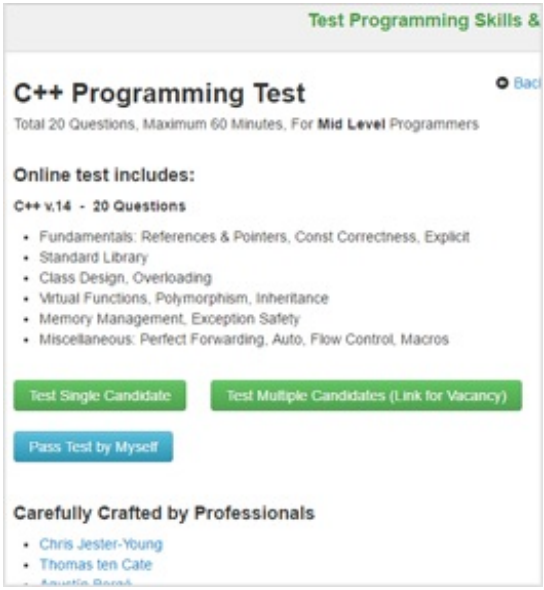
↑↑ Scroll up to the list of C++ questions

[BONUS – C++ Online Test]

Prefer a more hands-off approach to selecting candidates? No problem: just use our automated multiple-choice questions **C++ test** instead. Just send prospective developers their own unique custom link, and you'll automatically receive an email notification if they pass.

You can make your candidate selection process even more efficient with a mix-and-match approach: use our test to **find the top**

candidates, and then bring them in for an **interview using our premade questions**. You can save time and effort by weeding out the majority of candidates before you ever see them in person!



↑↑ Scroll up to the list of C++ questions

Conclusion

It remains a tricky business to assess an interview candidate's worth within the space of an hour, or even two. Using a set of **well-tested c++ programming interview questions** like the above, and calibrating them by using them on many different candidates, will help you take some of the noise out of the equation to get a better signal on the candidate's abilities. This, in turn, will result in **better hiring decisions**, a **stronger team**, and eventually a **better-functioning organization**.

10 Proven C++ Programming Questions to Ask on Interview (Explanations, Possible Answers, Following Questions)Click To Tweet

↑↑ Scroll up to the list of C++ questions

Authors

These C++ technical interview questions have been created by this team of C++ professionals:

↑↑ Scroll up to the list of C++ questions

Do you want extra questions? Read more from Pangara's team.

Like the article? Share it please!

It will REALLY help us to make **more content** here.