CERTLABZ CERTIFICATION BOARD

Certified C++ Programmer (CPPP)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working C++ programmer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1 — Domain 1: C++ Fundamentals and Syntax — DATA

Emma Wilson maps each C++ operator to its role, with one role missing.

Operator	Role
::	Scope resolution
=	Assignment
<<	(to choose)
sizeof	Byte size of type

Table 1.1 Operators and roles.

In cout << x, what does << do here?

A. Bitwise shift only
B. Address of x
C. Stream insertion
D. Comparison test

Correct answer: C
When used with a stream such as cout, the << operator performs stream insertion and writes the value to the output. The same symbol also means bitwise left shift with integer operands, but in stream context it inserts data, not shifts bits.

Question 2 — Domain 2: Data Types and References — CONFIG

James Carter checks how the auto keyword deduces a type from an expression.

```
auto x = 5;
auto y = 3.5;
auto z = x + y;
```

Listing 2.1 Type deduction with auto.

What type does z hold?

A. unsigned int
B. double
C. float
D. char

Correct answer: B
Adding an int to a double promotes the int, so the result is a double and auto deduces z as double. The auto keyword takes the type of the initializing expression, which here is the wider floating type.

Question 3 — Domain 3: Control Flow and Functions — CONFIG

Olivia Davis traces a simple counting loop that accumulates a sum.

```
int s = 0;
for (int i = 1; i <= 4; ++i)
    s += i;
```

Listing 3.1 A counting loop.

What is the final value of s?

A. 6
B. 7
C. 4
D. 10

Correct answer: D
The loop runs for i equal to 1, 2, 3 and 4, adding each to s in turn. The running total is 1 plus 2 plus 3 plus 4, which equals 10 when the loop finishes.

Question 4 — Domain 4: Classes, Objects and Templates — CONFIG

Liam Anderson calls a virtual function through a base class pointer.

```
struct Base { virtual int f(){return 1;} };
struct Der : Base { int f(){return 2;} };
Base* p = new Der();
cout << p->f();
```

Listing 4.1 A virtual call.

What does this print?

A 2
B 1
C 0
D A compile error

Correct answer: A
Because f is virtual, the call through the base pointer uses the dynamic type of the object, which is Der. That override returns 2, so the program prints 2 rather than the base version.

Question 5

Domain 5: Memory Management and the STL

DATA

Ava Thompson maps each library type to its purpose, with one purpose missing.

Type	Purpose
unique_ptr	Sole ownership
shared_ptr	(to choose)
vector	Dynamic array
map	Key to value

Table 5.1 Library types and purpose.

What does shared_ptr provide?

A Manual free needed
B No ownership held
C Ref-counted owner
D Stack storage

Correct answer: C
A shared_ptr is a reference-counted owner that lets several pointers share one object and frees it when the last owner goes away. This differs from unique_ptr, which keeps sole ownership, and from a raw pointer that owns nothing.

Question 6

Domain 6: Error Handling, Testing and Best Practices

CONFIG

David Brown reviews a try block that throws and a catch that takes a base reference.

```
try {
    throw std::runtime_error("bad");
} catch (const std::exception& e) {
    cout << "caught";
}
```

Listing 6.1 Catching by base reference.

Why does the catch block run?

- A Exceptions are ignored
- B Base ref catches it
- C Throw is a warning
- D Catch runs always

Correct answer: B
A runtime_error derives from std::exception, so a catch that takes a const reference to the base type also catches the derived object. Catching by base reference is the common way to handle a family of exception types.

Question 7 Domain 3: Control Flow and Functions

A developer writes two functions with the same name but different parameter types, and the compiler picks the right one at each call. What C++ feature is this?

- A Virtual dispatch
- B Recursion
- C Type inference
- D Overloading

Correct answer: D
Defining several functions with the same name but different parameter lists is function overloading, and the compiler resolves each call by its arguments. Virtual dispatch chooses at run time by object type, while recursion and inference are unrelated ideas.

Question 8 Domain 4: Classes, Objects and Templates

A base class has a function meant to be replaced by each subclass and called through a base pointer so the right version runs. Which keyword makes that happen?

- A virtual
- B static const
- C inline
- D friend

Correct answer: A
Marking the base function virtual enables dynamic dispatch, so a call through a base pointer runs the overriding version in the actual object. The static, inline and friend keywords do not change which override is selected at run time.

Question 9

Domain 5: Memory Management and the STL

A class acquires a resource in its constructor and releases it in its destructor, so the resource is always freed when the object goes out of scope. What idiom is this?

A. Singleton
B. Manual cleanup
C. RAII
D. Garbage collection

Correct answer: C
Tying a resource lifetime to an object lifetime, acquiring in the constructor and releasing in the destructor, is the RAII idiom. It removes the need for manual cleanup and gives deterministic release without a garbage collector.

Question 10

Domain 6: Error Handling, Testing and Best Practices

A team returns error codes from every function, but callers keep forgetting to check them, so failures slip through unnoticed. Which mechanism forces a failure to be handled?

A. More comments
B. Exceptions
C. Global flags
D. Silent returns

Correct answer: B
An exception cannot be silently ignored: it propagates up the call stack until a handler deals with it or the program stops. That makes exceptions a stronger choice than error codes when a failure must not be quietly skipped.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CPPP-001. https://certlabz.com/certifications/certified-cpp-programmer.html