C++ Quiz

* Required

Questions

What is the output of the following C++ code?

```
#include <iostream>

int main() {
   int a = 5;
   int b = 2;
   std::cout << a / b << std::endl;
   return 0;
}
```

(1 Point) *

○ 0

● 2

○ 2.5

○ 3

What is the output of the following code snippet?

```
#include <iostream>

template <typename T>
T add(T a, T b) {
     return a + b;
}
int main() {
     std::cout << add(5, 3) << std::endl;
```

```
    return 0;
}
```

(1 Point) *

○ Compilation error

○ Garbage value

○ 0

● 8

Which of the following is true about exception handling in C++? (1 Point) *

○ Exceptions are handled using the try, handle, finally blocks.

● You can throw an exception of any type.

○ All exceptions must be caught by reference.

○ Only standard exceptions defined in <exception> can be thrown.

Which of the following containers in the Standard Template Library (STL) is implemented as a dynamic array? (1 Point) *

○ std::list

○ std::deque

○ std::set

● std::vector

Which of the following best describes the "Rule of Five" in C++11 and beyond? (1 Point) *

○ A principle that suggests a function should not have more than five parameters.

○ A rule that a class should not have more than five member variables.

● A guideline stating that if a class defines one of the copy constructor, copy assignment operator, move constructor, move assignment operator, or destructor, it should define all five.

○ A rule that enforces the use of five specific design patterns in C++.

Which of the following statements about the execution policies introduced in C++17 for algorithms is true? (1 Point) *

○ Execution policies replace the need for threading libraries.

● Execution policies guarantee that algorithms will run faster.

○ Execution policies allow algorithms to be executed concurrently or in parallel.

○ Execution policies are only applicable to sorting algorithms.

Which of the following statements is true regarding std::shared_ptr and std::unique_ptr in C++? (1 Point) *

○ std::shared_ptr uses reference counting and cannot be copied.

○ std::unique_ptr allows shared ownership of a resource.

● std::unique_ptr cannot be copied but can be moved.

○ std::shared_ptr does not manage the lifetime of a resource.

What is the result of the following code?

#include <iostream>

```
constexpr int compute(int x) {
    return x * x;
}

int main() {
    constexpr int val = compute(5);
    std::cout << val << std::endl;
    return 0;
}
```

(1 Point) *

○ 0

○ Compilation error due to constexpr function

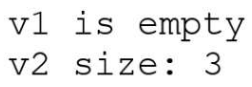
○ Undefined behavior

● 25

What will be the output of the following C++ code involving move semantics?

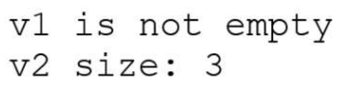
```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v1 = {1, 2, 3};
    std::vector<int> v2 = std::move(v1);

    if (v1.empty()) {
       std::cout << "v1 is empty" << std::endl;
    } else {
       std::cout << "v1 is not empty" << std::endl;
    }

   std::cout << "v2 size: " << v2.size() << std::endl;

   return 0;
}
```

(1 Point) *



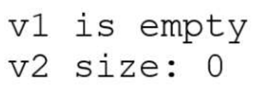
```
v1 is empty
v2 size: 3
```

● Option 1



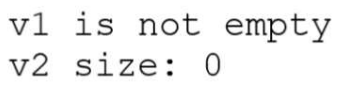
```
v1 is not empty
v2 size: 3
```

○ Option 2



```
v1 is empty
v2 size: 0
```

○ Option 3



```
v1 is not empty
v2 size: 0
```

○ Option 4

What will be the output of the following code involving the delete keyword in C++11?

```
#include <iostream>

class MyClass {
public:
    MyClass(int x) {}
    MyClass(double) = delete;
};

int main() {
    MyClass obj1(10);
    // MyClass obj2(3.14);
    std::cout << "Object created" << std::endl;
    return 0;
}
```

(1 Point) *

○ Runtime error due to deleted function

○ No output

○ Compilation error due to deleted constructor

● Object created

What is the output of the following C++ code?

```
#include <iostream>
struct Base {
     virtual void func(int x = 10) {
         std::cout << "Base: " << x << std::endl;
     }
 };

struct Derived : Base {
     void func(int x = 20) override {
         std::cout << "Derived: " << x << std::endl;
     }
};

int main() {
     Base* obj = new Derived();
     obj->func();
     delete obj;
     return 0;
}
```

(1 Point) *

○ Derived: 20

● Derived: 10

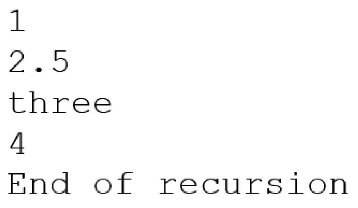
○ Base: 20

○ Base: 10

Consider the following C++ code using variadic templates:

```
#include <iostream>

void print() {
    std::cout << "End of recursion" << std::endl;
}

template<typename T, typename... Args>
void print(T first, Args... args) {
    std::cout << first << std::endl;
    print(args...);
}

int main() {
    print(1, 2.5, "three", 4);
    return 0;
}
```

What will be the output?
(1 Point) *



```
1
2.5
three
4
End of recursion
```

● Option 1

○ Compilation error due to type mismatch

○ Only the first argument is printed

○ Undefined behavior at runtime

What will be the output of the following C++ code involving templates and inheritance?

```
#include <iostream>
```

```
template<typename T>
class Base {
public:
     void func() {
        static_cast<T*>(this)->impl();
     }
  };

class Derived : public Base<Derived> {
public:
     void impl() {
        std::cout << "Derived implementation" << std::endl;
     }
};

int main() {
     Derived d;
     d.func();
     return 0;
}
```

(1 Point) *

○ Runtime error due to invalid cast

○ Compilation error due to static_cast

○ Infinite recursion

● Derived implementation

What is the main difference between std::atomic and volatile in C++? (1 Point) *

○ std::atomic is a C++11 feature, while volatile has been deprecated.

○ Both are used for multithreaded synchronization.

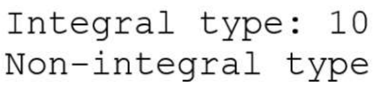
● std::atomic provides atomic operations suitable for multithreading, whereas volatile indicates that a variable may be modified outside the program flow.

○ volatile ensures atomic operations, while std::atomic does not.

What will be the output of the following code involving constexpr and if constexpr in C++17?

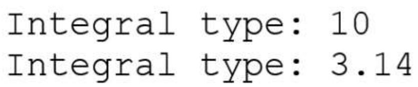
```
#include <iostream>

template<typename T>
void func(T value) {
    if constexpr (std::is_integral_v<T>) {
        std::cout << "Integral type: " << value << std::endl;
    } else {
        std::cout << "Non-integral type" << std::endl;
    }
}

int main() {
    func(10);
    func(3.14);
    return 0;
}
```

(1 Point) *



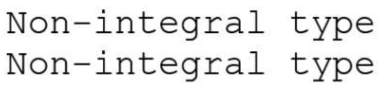
```
Integral type: 10
Non-integral type
```

● Option 1



```
Integral type: 10
Integral type: 3.14
```

○ Option 2



```
Non-integral type
Non-integral type
```

○ Compilation error due to type mismatch

○ Option 4

Page 2 of 2