WEEK 41

**# Topics**

**## Mini Lectures**

- Mini Lecture 1: Polymorphism

  - Compile-time Polymorphism e.g Operator overloading

  - Runtime Polymorphism & Virtual Functions

- Mini Lecture 2: Exceptions

- Mini Lecture 3: Algorithms & Big O notation

**## Timeline**

> This is an interactive lecture and the timeline is TENTATIVE.

- Mini Lecture 1

- Group Activity

- Mini Lecture 2

- Group Activity

- Lunch break

- Mini Lecture

- Group Activity

- Recap

// ---------------------------------------------------------------------------------------

**# Homework**

Total estimated workload is  about 1 hours 40min:

> Please choose one option from the followings:

1. Beginner friendly:

- [Polymorphism](<https://www.w3schools.com/cpp/cpp_polymorphism.asp>)

- [Exceptions](<https://www.w3schools.com/cpp/cpp_exceptions.asp>)

2. C++ from Beginner to Advanced (30 hour). We will cover:

- Chapter 19: Polymorphism  @26:21:03

[Link](<https://www.youtube.com/watch?v=8jLOx1hD3_o>)

// ---------------------------------------------------------------------------------------------------

**# Exam Related**

**## About:**

> There are two 2 tracks. You can choose ONLY one option

- Date: MWednesday (26/10/2022)

- Time: 10:00 - 13:00

- Duration: 150 min

- Place: online, same Zoom link

> If you are not comfortable using the webcam, you have the possibility to take the exam in Espoo's Campus. Please let me know before 13:00 today (14/10/2022).

> Please ensure that you have the basic tools in place:

- A Web camera

- Microphone

**## Practice for Pass / Fail track**

- [Exercises](<https://www.w3schools.com/cpp/exercise.asp?filename=exercise_syntax1>)

- [Quiz](<https://www.w3schools.com/quiztest/quiztest.asp?qtest=CPP>)

- [Review](<https://www.w3schools.com/cpp/>)

**## Review material**

- [Video 4h](<https://youtu.be/vLnPwxZdW4Y>)

- Lecture material

- Group activities

- [4 basic concepts of object-oriented programming](<https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming>)

- [The 4 Main Blocks Of Object-Oriented Programming](<https://www.apollotechnical.com/why-object-oriented-programming-matters/>)

# C++ Quiz Results

Score: 23 of 25

### 92% Correct:

### Question 1:

What is a correct syntax to output "Hello World" in C++?

cout << "Hello World";    Your answer

Console.WriteLine("Hello World");

System.out.println("Hello World");

print ("Hello World");

### Question 2:

C++ is an alias of C#

False    Your answer

True

### Question 3:

How do you insert COMMENTS in C++ code?

// This is a comment    Your answer

# This is a comment

/\* This is a comment

### Question 4:

Which data type is used to create a variable that should store text?

string    Your answer

Txt

myString

String

### Question 5:

How do you create a variable with the numeric value 5?

int x = 5;    Your answer

num x = 5

x = 5;

double x = 5;

### Question 6:

How do you create a variable with the floating number 2.8?

double x = 2.8;    Your answer

byte x = 2.8

x = 2.8;

int x = 2.8;

### Question 7:

Which method can be used to find the length of a string?

length()    Your answer

len()

getSize()

getLength()

### Question 8:

Which operator is used to add together two values?

The + sign    Your answer

The \* sign

The & sign

### Question 9:

The value of a string variable can be surrounded by single quotes.

False    Your answer

True

### Question 10:

Which header file lets us work with input and output objects?

#include <iostream>    Your answer

#include <inputstr>

#include <iosstring>

#include <stream>

### Question 11:

Which operator can be used to compare two values?

==    Your answer

<>

><

=

### Question 12:

To declare an array in C++, define the variable type with:

[]    Your answer

()

{}

### Question 13:

Array indexes start with:

0    Your answer

1

### Question 14:

How do you create a function in C++?

functionName()    Your answer

functionName.

(functionName)

functionName[]

### Question 15:

How do you call a function in C++?

functionName();    Your answer

functionName[];

functionName;

(functionName);

### Question 16:

Which keyword is used to create a class in C++?

class()    Your answer

className

class    Correct answer

MyClass

### Question 17:

What is the correct way to create an object called myObj of MyClass?

class MyClass = new myObj();    Your answer

class myObj = new MyClass();

new myObj = MyClass();

MyClass myObj;    Correct answer

### Question 18:

In C++, it is possible to inherit attributes and methods from one class to another.

True    Your answer

False

### Question 19:

Which method can be used to find the highest value of x and y?

max(x,y)    Your answer

largest(x,y)

maximum(x,y)

maxNum(x,y)

### Question 20:

Which operator is used to multiply numbers?

\*    Your answer

%

#

x

### Question 21:

How do you create a reference variable of an existing variable?

With the & operator    Your answer

With the ref word

With the REF word

With the \* operator

### Question 22:

How do you start writing an if statement in C++?

if (x > y)    Your answer

if x > y then:

if x > y:

### Question 23:

How do you start writing a while loop in C++?

while (x > y)    Your answer

x > y while {

while x > y {

while x > y:

### Question 24:

Which keyword is used to return a value inside a method?

return    Your answer

void

get

break

### Question 25:

Which statement is used to stop a loop?

break    Your answer

return

exit

stop

#### [Study C++ in our C++ Tutorial](https://www.w3schools.com/cpp/default.asp)

// ---------------------------------------------------------------------------------------------------------------

**# Group activity 1**

> The link to the online Compiler is below.

Exercise 1a - 1f from the [learncpp site](<https://www.learncpp.com/cpp-tutorial/virtual-functions/>)

**## Links**

3. [Learncpp](<https://www.learncpp.com/cpp-tutorial/the-stack-and-the-heap/>)

4. [Online Compiler](<https://cpp.sh/>)

#include <iostream>

#include <string\_view>

class Base

{

public:

virtual std::string\_view getName() const { return "Base"; } // note addition of virtual keyword

};

class Derived: public Base

{

public:

virtual std::string\_view getName() const { return "Derived"; }

};

int main()

{

Derived derived;

Base& rBase{ derived };

std::cout << "rBase is a " << rBase.getName() << '\n';

return 0;

}

we get:

rBase is a Derived

// -----------------------------------------------------

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

virtual std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

virtual std::string\_view getName() const { return "C"; }

};

class D: public C

{

public:

virtual std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

A& rBase{ c };

std::cout << "rBase is a " << rBase.getName() << '\n';

return 0;

}

What do you think this program will output?

Let’s look at how this works. First, we instantiate a C class object. rBase is an A reference, which we set to reference the A portion of the C object. Finally, we call rBase.getName(). rBase.getName() evaluates to A::getName(). However, A::getName() is virtual, so the compiler will call the most-derived match between A and C. In this case, that is C::getName(). Note that it will not call D::getName(), because our original object was a C, not a D, so only functions between A and C are considered.

As a result, our program outputs:

rBase is a C

rBase is a C

// -----------------------------------------------------------------------------------------------------------------------------

**A more complex example**

Let’s take another look at the Animal example we were working with in the previous lesson. Here’s the original class, along with some test code

#include <iostream>

#include <string>

#include <string\_view>

class Animal

{

protected:

std::string m\_name;

// We're making this constructor protected because

// we don't want people creating Animal objects directly,

// but we still want derived classes to be able to use it.

Animal(const std::string& name)

: m\_name{ name }

{

}

public:

const std::string& getName() const { return m\_name; }

std::string\_view speak() const { return "???"; }

};

class Cat: public Animal

{

public:

Cat(const std::string& name)

: Animal{ name }

{

}

std::string\_view speak() const { return "Meow"; }

};

class Dog: public Animal

{

public:

Dog(const std::string& name)

: Animal{ name }

{

}

std::string\_view speak() const { return "Woof"; }

};

void report(const Animal& animal)

{

std::cout << animal.getName() << " says " << animal.speak() << '\n';

}

int main()

{

Cat cat{ "Fred" };

Dog dog{ "Garbo" };

report(cat);

report(dog);

return 0;

}

Fred says ???

Garbo says ???

// -----------------------------------------------------------------------------------------------------------------------------------

**Quiz time**

1. What do the following programs print? This exercise is meant to be done by inspection, not by compiling the examples with your compiler.

1a)

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

virtual std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

// Note: no getName() function here

};

class D: public C

{

public:

virtual std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

A& rBase{ c };

std::cout << rBase.getName() << '\n';

return 0;

}

B. rBase is an A reference pointing to a C object. Normally rBase.getName() would call A::getName(), but A::getName() is virtual so it instead calls the most derived matching function between A and C. That is B::getName(), which prints B.

It prints B

B

// ------------------------------------------------------------------------------------------------------------------------------------

1b)

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

virtual std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

virtual std::string\_view getName() const { return "C"; }

};

class D: public C

{

public:

virtual std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

B& rBase{ c }; // note: rBase is a B this time

std::cout << rBase.getName() << '\n';

return 0;

}

It prints C

C. This is pretty straightforward, as C::getName() is the most derived matching call between classes B and C.

// -----------------------------------------------------------------------------------------------------------------------------------

1c)

#include <iostream>

#include <string\_view>

class A

{

public:

// note: no virtual keyword

std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

virtual std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

virtual std::string\_view getName() const { return "C"; }

};

class D: public C

{

public:

virtual std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

A& rBase{ c };

std::cout << rBase.getName() << '\n';

return 0;

}

It prints A

A. Since A is not virtual, when rBase.getName() is called, A::getName() is called

// -----------------------------------------------------------------------------------------------------------------------------------

1d)

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

// note: no virtual keyword in B, C, and D

std::string\_view getName() const { return "B"; }

};

class C: public B

{

public:

std::string\_view getName() const { return "C"; }

};

class D: public C

{

public:

std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

B& rBase{ c }; // note: rBase is a B this time

std::cout << rBase.getName() << '\n';

return 0;

}

It prints C

C. Even though B and C aren’t marked as virtual functions, A::getName() is virtual and B::getName() and C::getName() are overrides. Therefore, B::getName() and C::getName() are considered implicitly virtual, and thus the call to rBase.getName() resolves to C::getName(), not B::getName()

// -----------------------------------------------------------------------------------------------------------------------------------

1e)

#include <iostream>

#include <string\_view>

class A

{

public:

virtual std::string\_view getName() const { return "A"; }

};

class B: public A

{

public:

// Note: Functions in B, C, and D are non-const.

virtual std::string\_view getName() { return "B"; }

};

class C: public B

{

public:

virtual std::string\_view getName() { return "C"; }

};

class D: public C

{

public:

virtual std::string\_view getName() { return "D"; }

};

int main()

{

C c;

A& rBase{ c };

std::cout << rBase.getName() << '\n';

return 0;

}

It prints A

1. This one is a little trickier. rBase is an A reference to a C object, so rBase.getName() would normally call A::getName(). But A::getName() is virtual, so it calls the most derived version of the function between A and C. And that is A::getName(). Because B::getName() and c::getName() are not const, they are not considered overrides! Consequently, this program prints A.

// ----------------------------------------------------------------------------------------------------------------------------------

1f)

#include <iostream>

#include <string\_view>

class A

{

public:

A() { std::cout << getName(); } // note addition of constructor

virtual std::string\_view getName() const { return "A"; }

};

class B : public A

{

public:

virtual std::string\_view getName() const { return "B"; }

};

class C : public B

{

public:

virtual std::string\_view getName() const { return "C"; }

};

class D : public C

{

public:

virtual std::string\_view getName() const { return "D"; }

};

int main()

{

C c;

return 0;

}

It prints A

1. Another tricky one. When we create a C object, the A part is constructed first. When the A constructor is called to do this, it calls virtual function getName(). Because the B and C parts of the class aren’t set up yet, this resolves to A::getName().

// ------------------------------------------------------------------------------------------------------------------------------------

// not deeded???

**# Group activity 2**

> We will have pair-programming only in Full Stack course i.e (ma,ti,to,pe). All C++ activities will be carried in groups. The link to the online Compiler is below.

**## Exercise 1**

Is there a rule of thumb to follow when deciding to use exceptions instead of asserts or vice versa. [Ref 1-2 below]

**## Exercise 2**

...

**## Ref**

1. [Ref 1](<https://stackoverflow.com/questions/409794/exception-vs-assert>)
2. [Ref 2](<https://stackoverflow.com/questions/8087895/c-error-codes-vs-asserts-vs-exceptions-choices-choices>)
3. [Learncpp](<https://www.learncpp.com/cpp-tutorial/>)
4. [Online Compiler](<https://cpp.sh/>)