

# Introduction to Modern C++ Course Outline

Ryan Baker

January 3, 2025

## Week 1: Introduction to C++

<b>1</b>	<b>Course Introduction</b>	<b>2</b>
1.1	Lecture Series Overview . . . . .	2
1.2	Why Learn C++ . . . . .	2
<b>2</b>	<b>Features of C++</b>	<b>2</b>
2.1	Evolution of C++ . . . . .	2
2.2	The C++ Philosophy . . . . .	3
2.3	C++ vs. Other Languages . . . . .	3
<b>3</b>	<b>Environment Setup</b>	<b>3</b>
3.1	Tools Required . . . . .	3
3.1.1	Text Editor . . . . .	3
3.1.2	Compiler . . . . .	4
3.2	“Hello, World!” Example . . . . .	4
<b>4</b>	<b>Basic Syntax and Structure</b>	<b>4</b>
4.1	Basic Structure of a C++ Program . . . . .	4
4.1.1	<code>int</code> <code>main()</code> . . . . .	4
4.1.2	Statements and Expressions . . . . .	5
4.2	Foundational Concepts . . . . .	5
4.2.1	Semicolons, <code>/* comments */</code> , and Whitespace . . . . .	5
4.2.2	Line-by-Line Execution . . . . .	5
4.3	Input and Output with <code>iostream</code> . . . . .	6
<b>5</b>	<b>Datatypes and Variables</b>	<b>6</b>
5.1	<code>sizeof</code> Operator . . . . .	6
5.2	Primitive Types . . . . .	7
5.3	Declaration and Definition . . . . .	8
5.3.1	Assignment Operator <code>=</code> . . . . .	8
5.3.2	Brace Initialization <code>{}</code> . . . . .	9
5.4	Arithmetic Operators . . . . .	9

## Week 2: How C++ Works

<b>1</b>	<b>The Build Process</b>	<b>2</b>
1.1	Source Code . . . . .	2
1.2	Preprocessor . . . . .	2
1.2.1	Text Substitution . . . . .	2
1.2.2	Conditional Compilation . . . . .	3
1.2.3	File Inclusion . . . . .	3
1.2.4	Preprocessor Output . . . . .	3
1.3	Compilation . . . . .	4
1.3.1	Compiler Output . . . . .	4
1.4	Linking . . . . .	4
<b>2</b>	<b>Introduction to Memory</b>	<b>5</b>
2.1	How C++ Uses Memory . . . . .	5
2.2	Pointers . . . . .	6
2.2.1	NULL Pointers . . . . .	7
2.2.2	Pointer Arithmetic . . . . .	7
2.2.3	Pointers to Pointers . . . . .	7

## Week 3: C++ Control Flow

<b>1</b>	<b>Functions</b>	<b>3</b>
1.1	Passing by Value vs. Passing by Reference . . . . .	3
1.2	Function Overloading . . . . .	4
<b>2</b>	<b>Scope</b>	<b>4</b>
2.1	Types of Scope . . . . .	4
2.1.1	Global Scope . . . . .	4
2.1.2	Local Scope . . . . .	4
2.1.3	Anonymous Scope . . . . .	4
2.2	Namespaces . . . . .	5
2.2.1	Namespace Operator :: . . . . .	5
2.2.2	using Namespaces . . . . .	5
<b>3</b>	<b>Conditions and Branching</b>	<b>5</b>
3.1	Boolean Expressions . . . . .	5
3.1.1	bool() Casts . . . . .	5
3.1.2	Comparison Operators: ==, !=, <, <=, >, >= . . . . .	5
3.1.3	Logical Operators: !, &&,    . . . . .	5
3.2	if Statements . . . . .	5
3.2.1	The Overhead of if Statements . . . . .	6
3.3	switch Statements . . . . .	6
3.4	Ternary Operator ? : . . . . .	6
<b>4</b>	<b>Loops</b>	<b>6</b>
4.1	while Loops . . . . .	6
4.1.1	do while Loops . . . . .	6
4.2	for Loops . . . . .	6
4.2.1	Blank Fields . . . . .	6
<b>5</b>	<b>Control Flow Keywords</b>	<b>6</b>
5.1	break Keyword . . . . .	6
5.2	continue Keyword . . . . .	6
5.3	return Keyword . . . . .	6

## Week 4: Introduction to Object-Oriented Programming

<b>1</b>	<b>Arrays</b>	<b>2</b>
1.1	Arrays and Pointers . . . . .	2
1.2	Multidimensional Arrays . . . . .	2
1.3	Array Initialization . . . . .	2
1.4	Dynamic Arrays . . . . .	2
<b>2</b>	<b>Structs</b>	<b>2</b>
2.1	Struct Initialization . . . . .	2
<b>3</b>	<b>Classes</b>	<b>2</b>
3.1	Constructors and Destructors . . . . .	2
3.1.1	Initializer Lists . . . . .	2
3.1.2	Default Initialization . . . . .	2
3.1.3	Copy Constructors . . . . .	2
3.2	Access Specifiers . . . . .	2
3.2.1	<code>private</code> Members . . . . .	2
3.2.2	<code>protected</code> Members . . . . .	2
3.2.3	<code>public</code> Members . . . . .	2
3.2.4	Structs vs. Classes . . . . .	2
3.3	<code>static</code> Members . . . . .	2

## Week 5: Advanced Object-Oriented Programming

<b>1</b>	<b>Principles of Object-Oriented Programming</b>	<b>2</b>
1.1	Abstraction . . . . .	2
1.2	Encapsulation . . . . .	2
1.3	Inheritance . . . . .	2
1.3.1	<code>virtual</code> Functions . . . . .	2
1.3.2	Interfaces . . . . .	2
1.4	Polymorphism . . . . .	2
1.5	Composition <code>// not usually included</code> . . . . .	2
<b>2</b>	<b>Operator Overloading</b>	<b>2</b>
2.1	Type Casting . . . . .	2
2.2	<code>friend</code> Functions . . . . .	2

## Week 6: The Standard Library

<b>1</b>	<b>Standard Containers</b>	<b>2</b>
1.1	Sequence Containers . . . . .	2
1.1.1	std::array . . . . .	2
1.1.2	std::vector . . . . .	2
1.1.3	std::deque . . . . .	2
1.1.4	std::list . . . . .	2
1.2	Associative Containers . . . . .	2
1.2.1	std::set . . . . .	2
1.2.2	std::map . . . . .	2
1.3	Unordered Containers . . . . .	2
1.3.1	std::unordered_set . . . . .	2
1.3.2	std::unordered_map . . . . .	2
1.4	std::sort . . . . .	2
1.5	std::find . . . . .	2
1.6	std::accumulate . . . . .	2
1.7	Container Adapters . . . . .	2
<b>2</b>	<b>Iterators</b>	<b>2</b>
<b>3</b>	<b>Ranges</b>	<b>2</b>
<b>4</b>	<b>Views</b>	<b>2</b>

## Week 7: Safety in C++

1	Undefined Behavior	2
2	Memory Safety	2
3	Smart Pointers	2
4	Exception Safety	2



## Week 8: Templates

<b>1</b>	<b>Introduction to Templates</b>	<b>2</b>
1.1	How Do Templates Work? . . . . .	2
<b>2</b>	<b>Function Templates</b>	<b>2</b>
2.1	Implicit Template Deduction . . . . .	2
2.2	Template Function Overloading . . . . .	2
2.3	Function Template Specialization . . . . .	2
<b>3</b>	<b>Class Templates</b>	<b>2</b>
3.1	Template Instantiation . . . . .	2
3.2	Class Template Specialization . . . . .	2
<b>4</b>	<b>Non-Type Template Parameters</b>	<b>2</b>
<b>5</b>	<b>Variadic Templates</b>	<b>2</b>

## **Week 9: Compile-Time Programming**

<b>1</b>	<b>Lambdas</b>	<b>2</b>
<b>2</b>	<b>Compile-Time Programming</b>	<b>2</b>
<b>3</b>	<b>Template Metaprogramming</b>	<b>2</b>