Introduction to Modern C++ Course Outline

Ryan Baker

January 3, 2025

Week 1: Introduction to C++

1 Course Introduction **2**
1.1 Lecture Series Overview . . . 2
1.2 Why Learn C++ . . . 2

2 Features of C++ **2**
2.1 Evolution of C++ . . . 2
2.2 The C++ Philosophy . . . 3
2.3 C++ vs. Other Languages . . . 3

3 Environment Setup **3**
3.1 Tools Required . . . 3
3.1.1 Text Editor . . . 3
3.1.2 Compiler . . . 4
3.2 "Hello, World!" Example . . . 4

4 Basic Syntax and Structure **4**
4.1 Basic Structure of a C++ Program . . . 4
4.1.1 `int main()` . . . 4
4.1.2 Statements and Expressions . . . 5
4.2 Foundational Concepts . . . 5
4.2.1 Semicolons, `/* comments */`, and Whitespace . . . 5
4.2.2 Line-by-Line Execution . . . 5
4.3 Input and Output with `iostream` . . . 6

5 Datatypes and Variables **6**
5.1 `sizeof` Operator . . . 6
5.2 Primitive Types . . . 7
5.3 Declaration and Definition . . . 8
5.3.1 Assignment Operator `=` . . . 8
5.3.2 Brace Initialization `{}` . . . 9
5.4 Arithmetic Operators . . . 9

Week 2: How C++ Works

1 The Build Process **2**
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

2 Introduction to Memory **5**
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

3 Memory Layout **8**
3.1 Text Segment . . . 8
3.2 Static Memory . . . 8
3.3 The Heap . . . 8

Week 3: C++ Control Flow

1 Functions **2**
- 1.1 Passing by Value vs. Passing by Reference . . . 2
- 1.2 Function Overloading . . . 3

2 Scope **3**
- 2.1 Types of Scope . . . 3
  - 2.1.1 Global Scope . . . 3
  - 2.1.2 Local Scope . . . 3
  - 2.1.3 Anonymous Scope . . . 3
- 2.2 Namespaces . . . 4
  - 2.2.1 Namespace Operator `::` . . . 4
  - 2.2.2 `using` Namespaces . . . 4

3 Conditions and Branching **4**
- 3.1 Boolean Expressions . . . 4
  - 3.1.1 `bool()` Casts . . . 4
  - 3.1.2 Comparison Operators: `==`, `!=`, `<`, `<=`, `>`, `>=` . . . 4
  - 3.1.3 Logical Operators: `!`, `&&`, `||` . . . 4
- 3.2 `if` Statements . . . 4
  - 3.2.1 The Overhead of `if` Statements . . . 5
- 3.3 `switch` Statements . . . 5
- 3.4 Ternary Operator `? :` . . . 5

4 Loops **5**
- 4.1 `while` Loops . . . 5
  - 4.1.1 `do while` Loops . . . 5
- 4.2 `for` Loops . . . 5
  - 4.2.1 Blank Fields . . . 5

5 Control Flow Keywords **5**
- 5.1 `break` Keyword . . . 5
- 5.2 `continue` Keyword . . . 5
- 5.3 `return` Keyword . . . 5

Week 4: Introduction to Object-Oriented Programming

1 Arrays **2**

1.1 Arrays and Pointers 2

1.2 Multidimensional Arrays 2

1.3 Array Initialization 2

1.4 Dynamic Arrays 2

2 Structs **2**

2.1 Struct Initialization 2

3 Classes **2**

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 `private` Members 2

3.2.2 `protected` Members 2

3.2.3 `public` Members 2

3.2.4 Structs vs. Classes 2

3.3 `static` Members 2

Week 5: Advanced Object-Oriented Programming

1 Principles of Object-Oriented Programming 2

1.1 Abstraction 2

1.2 Encapsulation 2

1.3 Inheritance 2

1.3.1 `virtual` Functions 2

1.3.2 Interfaces 2

1.4 Polymorphism 2

1.5 Composition `// not usually included` 2

2 Operator Overloading 2

2.1 Type Casting 2

2.2 `friend` Functions 2

Week 6: The Standard Library

1 Standard Containers **2**

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

2 Iterators **2**

3 Ranges **2**

4 Views **2**

Week 7: Safety in C++

1 Undefined Behavior **2**

2 Memory Safety **2**

3 Smart Pointers **2**

4 Exception Safety **2**

Week 8: Templates

1 Introduction to Templates **2**

1.1 How Do Templates Work? 2

2 Function Templates **2**

2.1 Implicit Template Deduction 2

2.2 Template Function Overloading 2

2.3 Function Template Specialization 2

3 Class Templates **2**

3.1 Template Instantiation 2

3.2 Class Template Specialization 2

4 Non-Type Template Parameters **2**

5 Variadic Templates **2**

Week 9: Compile-Time Programming

1 Lambdas **2**

2 Compile-Time Programming **2**

3 Template Metaprogramming **2**