Introduction and Setup

Introduction to Modern C++

Ryan Baker

February 8, 2025

Lecture Objectives

- To understand C++'s features at a very high level.
- To install necessary developer tools.
- To understand the basic flow of a C++ program.
- To understand the concepts of datatypes, variables, and how to work with them in C++.
- To become acquainted with the `iostream` library for basic I/O.

Contents

1 Introduction to C++

C++ is a *general purpose* programming language developed by Bjarne Stroustrup in 1979 at Bell Labs. In its nearly 50 years in existence, it has evolved from a simple extension of C into a powerful and modern programming language.

1.1 Why C++?

"Why C++?" can be broken into two separate questions: "Why *learn* C++?" and "Why *use* C++?" The former has a much more decided answer than the latter, largely because the question "Why *use* C++?" is context dependent.

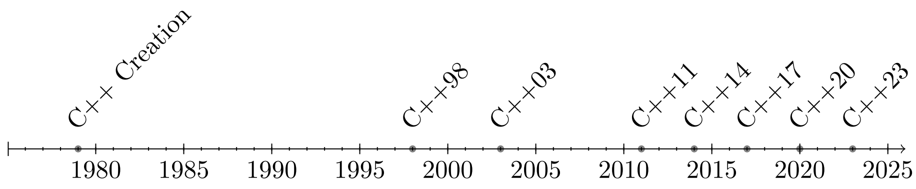
Why Learn C++?

1. C++ has a very strong "knowledge passport". Understanding C++ translates well to ability to learn new languages and programming concepts.
2. C++, more than any other language, is the one language that can do it all. See §1.3 for an elaboration.
3. Industry demand for C++ *may* skyrocket in the wake of young developers taking up new languages. Or maybe not. I am not an oracle.

Why Use C++?

1. Performance. C++ is a performant language. Only fools argue this.
2. Freedom. C++ provides a great deal of freedom to the programmer through manual memory management and low-level access.
3. Standardized. C++ is a standardized language, allowing the programmer to make certain assumptions about program behavior.

1.2 Evolution of C++



C++ has evolved significantly since its creation, adapting to modern programming needs while maintaining backward compatibility with C. Since 2011, C++ has seen an update every 3 years, with at least two more expected in 2026 and 2029.

C++'s evolution has been a great source of controversy. Because it has to maintain compatibility with C, some language features are more cumbersome in C++ than they would be in other languages. On the other hand, C++'s evolution has turned it into, arguably, the most powerful programming language available.

1.3 C++ vs. Other Languages

C++ distinguishes itself from other programming languages through its unique combination of features and design choices that make it powerful and versatile.

Compiled. C++ is a compiled language, meaning its source code is translated directly into machine code. The main alternative to compiled languages are interpreted languages, such as Python or JavaScript, which get executed line by line at runtime. Other examples of compiled languages are C, Zig, Rust, and Go. Compiled languages tend to be faster at execution time but have a more involved developer experience.

Static typing. C++ enforces type rules at compile time, helping to prevent "runtime surprises". The alternative to static typing is dynamic typing, where values are given types at runtime.

Multi-paradigm. It is not correct to call C++ an object-oriented language. Rather, C++ supports object oriented programming. C++ also supports other programming paradigms such as functional and procedural.

2 Environment Setup

2.1 Tools Required

To develop C++, two basic tools are required: a **text editor** and a **compiler**.

2.1.1 Text Editor

A text editor is a tool to edit plain text `/* duh */`. Text editors can be very basic, however, often, developers prefer to work with more comprehensive tools such as IDEs. An IDE (Integrated Development Environment) is a tool that includes a text editor as well as language specific debugging and development capabilities. Some popular IDEs for C++ include:

- **VSCode**: A very popular and free IDE with support for C/C++ through extensions. VSCode has no shortage of plugins and customizations.
- **CLion**: A modern IDE from JetBrains that supports advanced code analysis and debugging across multiple platforms.

If you prefer a lighter weight development experience, you should consider using a text editor such as Vim or Emacs.

2.1.2 Compiler

The compiler is responsible for translating your C++ code into machine-readable instructions. Some popular compilers include:

- **clang**: My personal choice and best for MacOS.
- **gcc**: The GNU Compiler Collection, best for Windows and Linux.
- **MSVC**: An increasingly irrelevant piece of garbage.

Throughout the lecture series, I will be using **clang**. If a certain flag or directive does not work for your compiler, simply look up its equivalent.

2.2 `"Hello, World!"` Example

With your text editor of choice, write the following code into `helloworld.cpp`:

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Compile `helloworld.cpp` using the following command for clang:

```
$ clang++ helloworld.cpp
```

or for gcc:

```
$ g++ helloworld.cpp
```

These commands will produce `a.out`, an executable file. Run the executable:

```
$ ./a.out
Hello, World!
```

You should see the desired message printed to the console.

3 Basic Syntax and Structure

3.1 `int main()`

The `main()` function serves a special purpose in a C/C++ program. Namely, it signifies the *entry point* for a program. When you run an executable, execution begins at `main()` and continues downward line-by-line.

The return value for `main()` is of type `int`. The integer value returned from main informs the system if the program executed successfully. `return 0` indicates successful execution, while a non-zero `return` indicates some type of failure.

```
int main()
{
    return 0; // exit success
}
```

Compiling and running the above program should will an exit success. The following program will yield an exit failure:

```
int main()
{
    return 1; // exit failure
}
```

One final `/* and rather unimportant */` note: C++ will insert a `return 0;` implicitly at the end of the `main()` function if none is written. This is only true for the `main()` function. Thus, the minimal C++ program is:

```
int main() {}
```

3.2 Semicolons, `/* comments */`, and Whitespace

Semicolons Every statement in C++ must end with a semicolon. Semicolons are what tell the compiler that the line is finished:

```
int x = 5;
int y = 10 // error: missing ‘;’
int z = 20;
```

Comments Comments are a way of “taking notes” that can help you and other readers understand what a certain piece of code is intended to do. In C++, there are two ways of making comments:

```
// 1) Line comments
// A ‘//’ tells the compiler that everything until
// the end of this line is a comment

/* 2) Block comments */
// Everything within /*...*/ is a comment
```

Advice: Prefer line comments to block comments. Block comments can be problematic because you may include a `*/` where you didn’t mean to:

```
/*
Block comments are denoted by /*...*/
*/ // error: this line is not a comment
```

Whitespace C++ does not care about whitespace. The following four programs are thus all equivalent:

```
int main() { return 0; }
```

```
int main() {
    return 0;
}
```

```
int main()
{
    return 0;
}
```

```
int main (    ) // if your code looks like this then
             {    // you’ve probably messed up
return
0

;                }
```

4 Datatypes and Variables

All of programming, at the end of the day, is just manipulation and interpretation of data. For us humans, data comes in many forms: integers, text, etc. As far as a computer is concerned, however, data only comes in bits. Because of this, it can be said that the only *real* difference between datatypes in C++ is their size, or how many bits they occupy.

Bits A bit, short for binary digit, is the fundamental unit of information. It can take on the values `0` or `1` (**false** or **true**).

Bytes A byte is a term for 8 bits. Because each bit can hold 2 possible values, a byte can hold $2^8 = 256$ possible values. This is similar to how an 8 digit number can hold 10^8 possible values.

4.1 Primitive Types

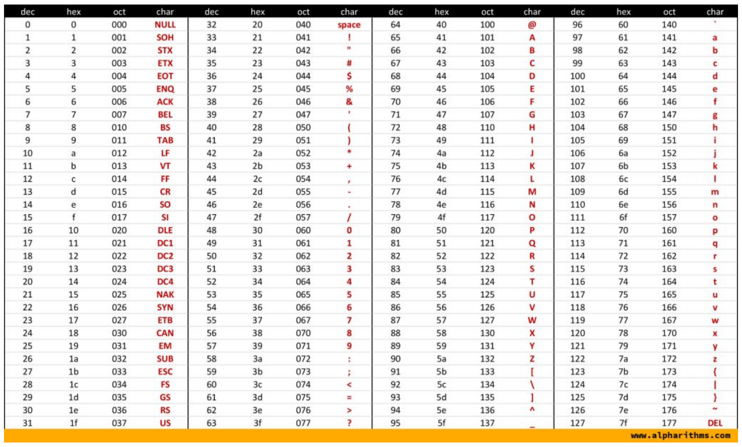
C++ provides a few built-in datatypes, often called primitive types:

- `int`: Represents integers (e.g., `0`, `1`, `42`, etc.).
- `char`: Represents characters (e.g., `'a'`, `'b'`, `'+'`, etc.).
- `bool`: Represents either `true` or `false`.
- `float`: Represents fractional numbers (e.g., `5.5`, `105.25`, etc.).
- `void`: Represents “no type”.

`int` The integer datatype typically occupies 4 bytes (32 bits) of memory, allowing it to represent integers from -2^{31} to $2^{31} - 1$. If, instead of representing negatives, you'd rather expand the range of representable positives, prepend the `unsigned` keyword to the integer:

```cpp
int x = 3000000000; // exceeds the bounds of int
unsigned int x = 3000000000; // works
```

`char` The character datatype is just an `int` in disguise. All characters are encoded as integers according to an ASCII table (shown below). Characters typically occupy 1 byte of memory, and can thus take on 256 possible values.



dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

www.alpharithms.com

The integer-character equivalence can be demonstrated:

```cpp
int x = ’a’;
std::cout << x << std::endl; // prints 97
```

```cpp
char x = 97;
std::cout << x << std::endl; // prints ’a’
```

`bool` The `bool` datatype is used to represent the logical values `true` or `false`. Although it only needs one bit in theory, it typically occupies 1 byte of memory. This is because computers have no efficient way of addressing individual bits.

`float` The `float` datatype is used to represent fractional values. It typically occupies 4 bytes of memory. If more precision or range is required, `double` exists and typically occupies 8 bytes of memory.

4.2 `sizeof` Operator

The `sizeof` operator allows us to get the size of a datatype or variable in bytes.

```cpp
std::cout << sizeof(int) << std::endl;
std::cout << sizeof(char) << std::endl;
std::cout << sizeof(bool) << std::endl;
// ...

int x = 42;
std::cout << sizeof(x) << std::endl;
```

4.3 Declaration and Definition

C++ variable initialization can be broken down into two pieces: **declaration** and **definition**. Variable declaration occurs when a variable is given a name and a type:

```cpp
int x; // "declares" an integer variable x
```

Definition occurs when a variable is assigned a value:

```cpp
x = 10; // "defines" x to be 10
```

Declaration and definition may occur on the same line:

```cpp
int x = 10; // "declares" and "defines" x to be 10
```

4.3.1 Assignment Operator =

The assignment operator assigns the value of the operand on its right to the variable on its left:

```cpp
x = 10; // assigns the value 10 to x
```

4.3.2 Brace Initialization {}

Brace initialization is a feature of modern C++ (C++11). The "standard" way to initialize a variable is using the = operator:

```cpp
int x = 42; // initializes x to 42
```

The shortcoming of this method has to do with *narrowing conversions*. A narrowing conversion occurs when a value does not fit into a datatype that it's assigned to, and thus has to be truncated or otherwise modified to fit. All of the following are examples of narrowing conversions:

```cpp
int x = 42.5;      // cannot store fractions
char y = 1000;     // sizeof char is 1 byte
unsigned z = -10; // z cannot represent negatives
```

These conversions are allowed by C++, mostly to maintain backwards compatibility with C. Brace initialization prevents these narrowing conversions by throwing errors when they occur:

```
int x { 42.5 };
char y { 1000 };
unsigned z { -10 };
```

4.4 Arithmetic Operators +, -, *, /, %

C++ supports the basic arithmetic operators for numerical datatypes:

```
int x = 10;
int y = 20;

int sum = x + y;  // sum = 30
int diff = x - y; // diff = -10
int prod = x * y; // prod = 200
int quot = y / x; // quot = 2
int mod = x % y   // mod = 0
```

Integer Division Dividing two integers often results in a non-integer value. Hence, in C++, truncation division is used:

```
std::cout << (20 / 3) << std::endl; // 6
std::cout << ( 1 / 2) << std::endl; // 0
std::cout << (-1 / 2) << std::endl; // 0
```

To get around this, you can use `float` types instead:

```
std::cout << (float(20)/3) << std::endl; // 6.66667
std::cout << (1.0 / 2) << std::endl; // 0.5
std::cout << (-1 / 2.0) << std::endl; // -0.5
```

Modulus The modulus operator can be thought of as the remainder operator. It returns the remainder of performing integer division on it's operands:

```
std::cout << (20 % 3) << std::endl; // 2
std::cout << ( 9 % 3) << std::endl; // 0
std::cout << (10 % 6) << std::endl; // 4
```

5 Basic I/O with `iostream`

5.1 Output with `std::cout`

The `std::cout` stream allows you to display messages or values on the console. It is part of the `iostream` library and uses the *insertion* operator `<<`.

```cpp
std::cout << "Hello, World!" << std::endl;

// multiple insertions
std::cout << "Hello, " << "World!" << std::endl;

std::cout << "The answer is: " << 42 << std::endl;
```

`std::endl` moves the cursor to the next line and flushes the output buffer.

5.2 Input with `std::cin`

The `std::cin` stream allows you to read values from user input. It uses the *extraction* operator `>>`.

```cpp
int x; // declare x as an integer

// print a message
std::cout << "Enter a number: ";

// read x
std::cin >> x;

std::cout << std::endl << "x = " << x << std::endl;
```

`std::cin`, by default, skips whitespace when reading input. Multiple inputs can be chained together:

```cpp
int x, y, z;
std::cin >> x >> y >> z;
```

A Installing VSCode and C++ Compiler

A.1 MacOS

1. **Install Visual Studio Code**
   - Go to the VSCode webpage.
   - Download the version for **macOS** and open the `.dmg` file.
   - Move VSCode into the Applications folder.
2. **Install Xcode Command Line Tools**
   - Open your Terminal and run the following command:

   ```
   $ xcode-select --install
   ```

   - Follow the prompts to install the Command Line Tools, which include **clang**.
3. **Verify Installation**
   - In your Terminal, type:

   ```
   $ clang --version
   ```

   If installed correctly, this command will print the version of clang.

A.2 Windows

1. **Consider booting your machine with Linux or buying a new one.**
2. **Install Visual Studio Code**
   - Go to the VSCode webpage.
   - Download the version for **Windows** and run the installer.
   - Follow the prompts to complete the installation.
3. **See Instruction 1**
4. **Install C++ Compiler with MinGW**
   - Download the MinGW-w64 installer from Mingw-w64 SourceForge.
   - Select architecture (32-bit or 64-bit) during installation.
   - Add the compiler to your system path:
     (a) Open the **Start Menu**, search for `Environment Variables`, and select **Edit the system environment variables**.

(b) In the **Environment Variables** window, click **Path**, then **Edit**.
(c) Click **New** and add the path to the `bin` directory of your MinGW installation (e.g., `C:\mingw-w64\bin`).

5. **Verify Installation**
   - In your terminal, type:

```
$ g++ --version
```

   If installed correctly, this command will print the version of g++.

6. **See Instruction 1**

A.3 Linux (Ubuntu/Debian-based)

If you are a Linux user then you're likely acquainted with a text editor such as Vim, Emacs, or Nano. Installing VSCode is likely not the best option for you.

1. **Install C++ Compiler (GCC)**
   - Install **g++** by running:

```
$ sudo apt install build-essential
```

2. **Verify Installation**
   - Check that g++ is installed by running:

```
$ g++ --version
```

B Setting Up VSCode for C++ Development

1. **Install C++ Extension**
   - Open VSCode and go to the **Extensions View** (`Ctrl+Shift+X` or `Cmd+Shift+X` or use the sidebar).
   - Search for **C/C++** by Microsoft and click **Install**.
2. **Configure Build Tasks**
   - Create a folder for you C++ project and open it in VSCode.
   - Create a `.cpp` file and write a program.
   - Open **Terminal**, and compile and execute your program.