Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Michael Steven Paucar Rojas

Introduction to C++

Lecture Notes

September 2025

PREFACE

Contents

Preface **2**

1 Basic Concepts of C++ **8**

1.1 Introduction 8

1.1.1 What is C++? 8

1.1.2 Why Use C++ 8

1.1.3 Difference between C and C++ 9

1.1.4 Compiled vs Interpreted Language 9

1.1.5 Is C++ Compiled or Interpreted? 9

1.1.6 Programming Paradigms in C++ 9

1.1.7 What Makes C++ More Robust? 10

1.1.8 Comparison with Python and Mathematica 10

1.2 Environment Setup 11

1.2.1 Installing C++ 11

1.2.2 Installing a C++ IDE 11

1.2.3 Personal Alternatives: VSCode and Eclipse 11

1.2.4 C++ Quick Start Guide 12

1.3 Syntax 13

1.3.1 Explained Example 14

1.3.2 Omitting the Namespace 15

1.3.3 Many Statements 16

1.4 Output (Printing Text) 16

1.4.1 Other Outputs 17

1.5 Output Numbers 18

1.6 New Lines 19
1.6.1 Other Escape Sequences 20
1.7 Comments 21
1.7.1 Single-line Comments 21
1.7.2 Multi-line Comments in C++ 22

2 Variables and Data Types 23

2.1 Variables 23
2.1.1 Declaring (Creating) Variables 23
2.1.2 Changing Variable Values 24
2.1.3 Other Types 24
2.1.4 Displaying Variables 25
2.1.5 Add Variables Together 25
2.2 Declare Multiple Variables 26
2.2.1 Declare Many Variables 26
2.2.2 One Value to Multiple Variables 26
2.3 Identifiers 26
2.4 Constants 27
2.4.1 Notes On Constants 27
2.5 Variables Examples 28
2.5.1 Real-Life Examples 28
2.5.2 Calculate the Area of a Rectangle 29
2.6 User Input 30
2.6.1 Complete <iostream>Reference 30
2.7 Data Types 30
2.7.1 Basic Data Types 31
2.8 Numeric Data Types 31
2.8.1 Numeric Types 31

2.8.2 Scientific Numbers 32
2.9 Boolean Data Types 32
2.9.1 Boolean Types 32
2.10 Character Data Types 33
2.10.1 Character Types 33
2.11 String Data Types 34
2.11.1 String Types 34
2.11.2 The auto Keyword 34
2.12 Data Types Examples 36
2.12.1 Real-Life Examples 36

3 Operators and Expressions 37
3.1 Operators 37
3.1.1 Arithmetic 37
3.1.2 Assignment 37
3.1.3 Comparison 37
3.1.4 Logical 37
3.1.5 Precedence 37
3.2 Strings 37
3.2.1 Concatenatio 37
3.2.2 Numbers and Strings 37
3.2.3 Strings Length 37
3.2.4 Access Strings 37
3.2.5 Special Characters 37
3.2.6 User Input Strings 37
3.2.7 Omitting Namespace 37
3.2.8 C-Style Strings 38
3.3 Math 38

3.3.1 Max and min 38
3.3.2 <cmath>Library 38
3.3.3 Complete Math Reference 38
3.4 Boolenas 38
3.4.1 Booleans 38
3.4.2 Boolean Expressions 38
3.4.3 Boolean Examples 38

4 Control Structures **39**
4.1 Decision Structures (Conditionals) 39
4.1.1 if 39
4.1.2 if...else 39
4.1.3 if..else if...else 39
4.1.4 switch 39
4.2 Repetition Structures (Loops) 39
4.2.1 for 39
4.2.2 while 39
4.2.3 do..while 39
4.3 Flow Control Within Loops 39
4.3.1 break 39
4.3.2 continue 39
4.3.3 return 40
4.4 Related Structures 40
4.4.1 Ternary Operator 40
4.4.2 goto 40

5 Functions **41**

6 Funciones **42**

7 Punteros y Referencias **43**

8 Estructuras **44**

9 Manejo de Archivos de Datos **45**

10 Análisis Numérico **46**

11 Programación Orientada a Objetos **47**

12 Reference **48**

Chapter 1

Basic Concepts of C++

1.1. Introduction

1.1.1 What is C++?

- C++ is a cross-platform language that can be used to create high-performance applications.
- C++ was developed by Bjarne Stroustrup, as an extension to the C language.
- C++ gives programmers a high level of control over system resources and memory.
- The language was updated 5 major times in 2011, 2014, 2017, 2020, and 2023 to C++11, C++14, C++17, C++20, and C++23.

1.1.2 Why Use C++

- C++ is one of the world's most popular programming languages.
- C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
- As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

1.1.3 Difference between C and C++

- C++ was developed as an extension of C, and both languages have almost the same syntax.
- The main difference between C and C++ is that C++ supports classes and objects, while C does not.

1.1.4 Compiled vs Interpreted Language

- **Compiled Language (C++, C, Rust):** The source code is fully translated into machine code before execution. It is faster and generates an executable.
- **Interpreted Language (Python, Mathematica):** The code is executed line by line during runtime. It is more flexible but slower.

Feature	Compiled	Interpreted
Execution	Pre-translation	Real-time translation
Speed	High	Medium to low
Output	Executable (`.exe`, `.out`)	No executable generated
Portability	Low	High (requires interpreter)

1.1.5 Is C++ Compiled or Interpreted?

C++ is a **compiled** language, which means it needs to go through a compiler (such as `g++`) to generate an executable before it can be run. This provides greater speed and efficiency, but less flexibility for rapid development.

1.1.6 Programming Paradigms in C++

C++ is a multiparadigm language. These are the main ones:

- **Imperative:** Sequential instructions to change the program's state.
- **Procedural:** Based on functions. Ideal for modular organization.
- **Object-Oriented Programming (OOP):** Use of classes, objects, inheritance, encapsulation, and polymorphism.

- **Generic:** Use of `templates` for reusable functions and classes.
- **Functional (limited):** Support for lambda functions, partial immutability.

1.1.7 What Makes C++ More Robust?

- **Full memory control:** You can use pointers, `new`, `delete`.
- **High performance:** Allows optimization close to the hardware level.
- **Multiparadigm:** Flexibility to use different programming styles.
- **Low-level access:** Can interact with hardware and operating systems.
- **Templates:** Efficient and reusable code generation.
- **Static compilation:** Errors are detected at compile time.
- **STL (Standard Template Library):** Ready-to-use collections, algorithms, and data structures.
- **Scalable:** Ideal for large, industrial, and mission-critical projects.

1.1.8 Comparison with Python and Mathematica

Feature	C++	Python	Mathematica
Type	Compiled	Interpreted	Interpreted
Paradigm	Multiparadigm	Multiparadigm	Functional - Symbolic
Speed	Very high	Medium	Low (symbolic)
Common use	Games, systems, hardware	Data science, scripting	Symbolic math, algebra

1.2. Environment Setup

1.2.1 Installing C++

If you want to run C++ on your own computer, you need two things:

- A text editor, such as Notepad, to write C++ code
- A compiler, such as GCC, to translate the C++ code into a language the computer can understand.
- There are many text editors and compilers to choose from. In the following steps, we will show you how to use an IDE that includes both.

1.2.2 Installing a C++ IDE

In this course, we will use the **Zinjal** development environment, a lightweight and free IDE specifically designed for C++ programming. Zinjal includes everything needed to get started:

- A code editor with syntax highlighting
- A preconfigured C++ compiler (based on g++)
- Basic integrated debugger
- Execution and analysis tools

You can download the latest version from (Zinjal).

> ⚠ If the program shows errors with accented characters (like tildes or ñ), make sure the source file is encoded in UTF-8. You can also check your system or compiler locale settings.

1.2.3 Personal Alternatives: VSCode and Eclipse

Personally, I use two more advanced development environments:

VSCode: is a modern, cross-platform code editor that allows comfortable C++ work thanks to its extensions. It requires installing a compiler (such as `MinGW`) and configuring tasks for compilation.

- Lightweight editor with integrated terminal and support for multiple languages.
- Useful extensions: Microsoft's `C/C++`, `Code Runner`, among others.
- Manual compilation or via tasks (`tasks.json`).

Eclipse CDT: (C/C++ Development Tooling) is a more robust environment aimed at large projects. It offers advanced integration with the compiler, debugger, and version control.

- Requires Java to run.
- Oriented to projects organized in folders with source files, headers, and build configuration.

Note: Take advantage of IDE features such as:

- **Custom snippets**: quick templates for common structures (like `main`, classes, loops, etc.)
- **Automated tasks**: configure build and run with a single click using `tasks.json`
- **CMake integration**: ideal for large, cross-platform projects
- **Visual debugger**: breakpoints, variable inspection, and step-by-step execution
- **IntelliSense / Autocomplete**: code suggestions, inline documentation, and quick navigation
- **Version control (Git)**: direct integration to manage your projects

1.2.4 C++ Quick Start Guide

The process of creating a new file in C++ is generally very similar across different development environments (IDEs). In most cases, it is enough

to create a new file and save the code with the `.cpp` extension. However, some environments may require additional steps, such as manually specifying the file type or programming language, or configuring a project before compilation is possible. Even so, the content and basic structure of a C++ program remains the same in all cases.

Let's create our first C++ file:

- Open the IDE and go to (File > New > Empty File).
- Type the following C++ code and save the file as *myfirstprogram.cpp* (File > Save File As):

```cpp
//myfirstprogram.cpp
#include <iostream>
using namespace std;

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

- Don't worry if you don't understand the above code; we will explain it in detail in later chapters. For now, focus on how to run it.
- Then, go to (Compile > Compile and Run) to execute the program. The output will be similar to this:

```
Hello World!
Process returned 0 (0x0) execution time : 0.011 s
Press any key to continue.
```

You have written and run your first C++ program.

1.3. Syntax

Let's break down the following code to understand it better:

```cpp
#include <iostream>
using namespace std;

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

1.3.1 Explained Example

- **Line 1:** `#include <iostream>` is a header file library that allows working with input and output objects, such as `cout` (used in line 5). Header files add functionality to C++ programs.
- **Line 2:** `using namespace std` means we can use names for objects and variables from the standard library.

> ⚠ Don't worry if you don't understand how `#include <iostream>` and `using namespace std` work. Think of them as something that (almost) always appears in your program.

- **Line 3:** A blank line. C++ ignores whitespace. But we use it to make the code more readable.
- **Line 4:** Another element that always appears in a C++ program is `int main()`. This is called a function. Any code inside its braces `"{ }"` will be executed.
- **Line 5:** `cout` (pronounced "see-out") is an object used together with the insertion operator (<<) to output/print text. In our example, it will output `"Hello World!"`.

> ⚠ **Note:**
>
> - C++ is case-sensitive: ‘‘`cout`’’ and ‘‘`Cout`’’ have different meanings.
> - Every C++ statement ends with a semicolon ‘‘`;`’’.
> - The `int main()` body could also be written as: `int main () { cout << "Hello World! "; return 0; }`
> - The compiler ignores whitespace. However, having mul-

tiple lines makes the code easier to read.

- **Line 6:** `return 0;` ends the main function.
- **Line 7:** Don't forget to add the closing brace `"}"` to finish the main function.

1.3.2 Omitting the Namespace

You may see some C++ programs running without the standard namespace library. This `using namespace std` line can be omitted and replaced with the `std` keyword followed by the `:: operator` for some objects:

```
1 #include <iostream>
2
3 int main() {
4   std::cout << "Hello World!";
5   return 0;
6 }
```

It is up to you whether you want to include the standard namespace library or not.

- A computer program is a list of "instructions" that must be "executed" by a computer.
- In a programming language, these programming instructions are called statements.
- The following statement "instructs" the compiler to print the text `''Hello World!''` on the screen:

```
1 cout << "Hello World!";
```

- It is important that you end the statement with a semicolon `";"`.
- If you forget the semicolon `(;)`, an error will occur and the program will not run:

```
1 cout << "Hello World!"  // ';' is not used
```

```
error: expected ';' before 'return'
```

1.3.3 Many Statements

Most C++ programs contain many statements.

Statements execute, one by one, in the same order they are written:

```
cout << "Hello World!";
cout << "Have a good day!";
return 0;
```

```
Hello World!
Have a good day!
```

- The first statement executes first (prints `Hello World!` on the screen).
- Then, the second statement executes (prints `Have a good day!` on the screen).
- And finally, the third statement executes (properly ends the C++ program).

1.4. Output (Printing Text)

The `cout` object, along with the (<<) operator, is used to output values and print text.

Just remember to surround the text with double quotes (“ ”).

```
#include <iostream>
using namespace std;

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

```
Hello World!
```

You can add as many `cout` objects as you want. However, keep in mind that a new line is not inserted at the end of the output:

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World!";
  cout << "I am learning C++";
  return 0;
}
```

```
Hello World!I am learning C++"
```

1.4.1 Other Outputs

Unlike other languages like Java, in C++ the standard error stream `cerr` does not print in red by default. To display errors in color (for example, red), it is necessary to include ANSI sequences like `\033[31m`, which change the text color in the console.

```cpp
cerr << "\033[31mIncorrect Option!!\033[0m";
```

```
Incorrect Option!!
```

Note: In C++, the `cerr` stream does not print in red by default. To do that, you need to add ANSI color codes manually.

Feature	`cout`	`cerr`
Channel	Standard output	Standard error
Buffering	Yes, **buffered**	No, prints immediately
Default color	Plain text (no color)	Plain text (no color)
Used for...	Normal output	Errors, warnings

- `cout`: Standard output stream, used to print text to the console.
- `cerr`: Output stream for errors, unbuffered, so messages appear immediately (useful for critical messages).

- `clog`: Similar to `cerr` but buffered, meaning it may delay printing for optimization. Used for logs or less urgent debug messages.
- `cin`: Standard input stream to read data from keyboard or standard input.

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Normal output\n";
    cerr << "Urgent error\n";
    clog << "Log message\n";
}
```

```
Normal output
Urgent error
Log message
```

1.5. Output Numbers

You can also use `cout` to print numbers.

However, unlike text, we do not put numbers inside double quotes.

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << 3;
  return 0;
}
```

```
3
```

You can also perform mathematical calculations:

```cpp
cout << 3 + 3;
```

```
6
```

1.6. New Lines

To insert a new line in your output, you can use the \n character:

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

```
Hello World!
I am learning C++
```

You can also use another << operator and place the \n character after the text, like this:

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World!" << "\n";
  cout << "I am learning C++";
  return 0;
}
```

```
Hello World!
I am learning C++
```

Note: Two \n\n characters one after the other will create a blank line:

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World!" << "\n\n";
  cout << "I am learning C++";
  return 0;
}
```

```
Hello World!

I am learning C++
```

Another way to insert a new line is with the `endl` manipulator:

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```

```
Hello World!
I am learning C++
```

- Both `\n` and `endl` are used to "break lines". However, `\n` is the most commonly used.
- The newline character (`\n`) is called an escape sequence and forces the cursor to move to the beginning of the next line on the screen.
- This generates a new line.

1.6.1 Other Escape Sequences

Escape Sequence	Description
`\t`	Creates a tab
`\\`	Inserts a backslash character (`\`)
`" "`	Inserts a double quote character

```
cout << "Hello World!\t";
cout << "I am learning C++";
```

```
Hello World! I am learning C++
```

```
cout << "Hello World!\\";
cout << "I am learning C++";
```

```
Hello World!\I am learning C++
```

```
cout << "They call him \"Michael\".";
```

```
They call him "Michael".
```

1.7. Comments

Comments can be used to explain C++ code and make it more readable. They can also be used to prevent execution when testing alternative code. Comments can be single-line or multi-line.

1.7.1 Single-line Comments

- Single-line comments start with two forward slashes (//).
- Any text between // and the end of the line is ignored by the compiler (it will not be executed).

This example uses a single-line comment before a line of code:

```
// This is a comment
cout << "Hello World!";
```

This example uses a single-line comment at the end of a line of code:

```
cout << "Hello World!"; // This is a comment
```

1.7.2 Multi-line Comments in C++

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by the compiler:

```
/* The code below will print the words Hello World!
to the screen, and it is amazing */
cout << "Hello World!";
```

Chapter 2

Variables and Data Types

2.1. Variables

Variables are containers for storing data values.

In C++, there are different types of variables (defined with different keywords), for example:

- `int` - stores whole numbers without decimals, like 123 or -123
- `double` - stores floating-point numbers with decimals, like 19.99 or -19.99
- `char` - stores single characters, like `'a'` or `'B'`. Char values are written between single quotes.
- `string` - stores text, like `"Hello world"`. String values are written between double quotes.
- `bool` - stores values with two states: true or false

2.1.1 Declaring (Creating) Variables

To create a variable, specify the type and assign a value:

```
type variableName = value;
```

Where type is one of the C++ types (such as int) and variableName is the variable's name (like x or myName). The equal sign is used to assign values to the variable.

To create a variable that should store a number, see the following example:

```
// Creates a variable named myNum of type int and assigns it
    the value 15:

int myNum = 15;
cout << myNum;
```

You can also declare a variable without assigning the value and assign it later:

```
int myNum;
myNum = 15;
cout << myNum;
```

2.1.2 Changing Variable Values

Note: that if you assign a new value to an existing variable, it will overwrite the previous value:

```
int myNum = 15;  // myNum is 15
myNum = 10;      // Now myNum is 10
cout << myNum;   // Outputs 10
```

2.1.3 Other Types

In C++, the basic types include `int` for whole numbers, `double` and `float` for decimal numbers, `char` for single characters, `string` for text, and `bool` for logical values. Additionally, there are larger integer types like `long` and `long long` for bigger integer values.

```
int myNum = 5;               // Integer (whole number without
    decimals)
double myFloatNum = 5.99;    // Floating point number (with
    decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

You will learn more about individual types in the Data Types section.

2.1.4 Displaying Variables

The `cout` object is used along with the << operator to display variables.

To combine text and a variable, separate them with the << operator:

```
int myAge = 27;
cout << "I am " << myAge << " years old.";
```

```
I am 27 years old.
```

You can also combine different types, which you will learn more about in a later section.

```
 #include <iostream>
#include <string>
using namespace std;

int main() {
  string name = "Michael";
  int age = 27;
  double height = 6.1;

  cout << name << " is " << age << " years old and " << height
      << " feet tall.";
  return 0;
}
```

```
Michael is 27 years old and 6.1 feet tall.
```

2.1.5 Add Variables Together

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

```
11
```

2.2. Declare Multiple Variables

2.2.1 Declare Many Variables

To declare more than one variable of the same type, use a comma-separated list:

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

```
61
```

2.2.2 One Value to Multiple Variables

You can also assign the same value to multiple variables in one line:

```
int x, y, z;
x = y = z = 50;
cout << x + y + z;
```

```
150
```

2.3. Identifiers

All C++ variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The general rules for naming variables are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case-sensitive (`myVar` and `myvar` are different variables)
- Names cannot contain whitespaces or special characters like `!`, `#`, `%`, etc.
- Reserved words (like C++ keywords, such as `int`) cannot be used as names

2.4. Constants

When you do not want others (or yourself) to change existing variable values, use the `const` keyword (this will declare the variable as "constant", which means **unchangeable and read-only**):

You should always declare the variable as constant when you have values that are unlikely to change:

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

You should always declare the variable as constant when you have values that are unlikely to change:

2.4.1 Notes On Constants

When you declare a constant variable, it must be assigned with a value:

Like this:

```cpp
const int minutesPerHour = 60;
```

This however, will **not work**:

```cpp
const int minutesPerHour;
minutesPerHour = 60; // error
```

```
prog.cpp: In function int main():
prog.cpp:5:13: error: uninitialized const minutesPerHour [-fpermissive]
    5 | const int minutesPerHour;
      | ~~~~~~~~~~~~~~
prog.cpp:6:18: error: assignment of read-only variable minutesPerHour
    6 | minutesPerHour = 60;
      | ~~~~~~~~~~~~~~~~~~^~~~
```

- `const` = can never be changed.
- Trying to reassign it causes a compilation error.
- It is useful to protect values that should not be modified.

2.5. Variables Examples

2.5.1 Real-Life Examples

Let's get a bit more practical!

Often in our examples, we simplify variable names to match their data type (myInt or myNum for `int` types, myChar for `char` types, and so on). This is done to avoid confusion.

However, for a practical example of using variables, we have created a program that stores different data about a college student:

```cpp
// Student data
int studentID = 15;
int studentAge = 23;
float studentFee = 75.25;
char studentGrade = 'B';

// Print variables
cout << "Student ID: " << studentID << "\n";
cout << "Student Age: " << studentAge << "\n";
cout << "Student Fee: " << studentFee << "\n";
cout << "Student Grade: " << studentGrade << "\n";
```

```
Student ID: 15
Student Age: 23
Student Fee: 75.25
Student Grade: B
```

2.5.2 Calculate the Area of a Rectangle

In this real-life example, we create a program to calculate the area of a rectangle (by multiplying the length and width):

```cpp
// Create integer variables
int length = 4;
int width = 6;

// Calculate the area of a rectangle
int area = length * width;

// Print the variables
cout << "Length is: " << length << "\n";
cout << "Width is: " << width << "\n";
cout << "Area of the rectangle is: " << area << "\n";
```

```
Length is: 4
Width is: 6
Area of the rectangle is: 24
```

2.6. User Input

You have already learned that `cout` is used to output (print) values. Now we will use `cin` to get user input.

`cin` is a predefined variable that reads data from the keyboard with the extraction operator (>>).

In the following example, the user can input a number, which is stored in the variable `x`. Then we print the value of `x`:

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

```
Type a number: 4
Your number is: 4
```

Note:

- `cout` is pronounced "see-out". Used for output, and uses the insertion operator (<<)
- `cin` is pronounced "see-in". Used for input, and uses the extraction operator (>>)

2.6.1 Complete <iostream>Reference

Tip: Both `cin` and `cout` belongs to the `<iostream>` library, which is short for standard *input / output streams*. For a complete reference of `<iostream>` objects along with detailed information, go to our C++ iostream Reference.

2.7. Data Types

As explained in the Variables chapter, a variable in C++ must be a specified data type:

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```

2.7.1 Basic Data Types

The data type specifies the size and type of information the variable will store:

Data Type	Size	Description
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values
int	2 or 4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6–7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

2.8. Numeric Data Types

2.8.1 Numeric Types

Use `int` when you need to store a whole number without decimals, like 35 or 1000, and `float` or `double` when you need a floating point number (with decimals), like 9.99 or 3.14515.

```
\\int
int myNum = 1000;
cout << myNum;
```

```
\\float
float myNum = 5.75;
cout << myNum;
```

```
\\ double
double myNum = 19.99;
cout << myNum;
```

Note (float vs. double):
The precision of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.

2.8.2 Scientific Numbers

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

```
  float f1 = 35e3;
  double d1 = 12E4;
  float f2 = 35e-3;
  double d2 = 12E-5;
```

```
35000
120000
0.035
0.00012
```

2.9. Boolean Data Types

2.9.1 Boolean Types

A boolean data type is declared with the bool keyword and can only take the values `true` or `false`.

When the value is returned, `true = 1` and `false = 0`.

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun;  // Outputs 1 (true)
cout << isFishTasty;  // Outputs 0 (false)
```

```
1
0
```

2.10. Character Data Types

2.10.1 Character Types

The `char` data type is used to store a single character. The character must be surrounded by single quotes, like `'A'` or `'c'`:

```
char myGrade = 'B';
cout << myGrade;
```

```
B
```

Alternatively, if you are familiar with ASCII, you can use ASCII values to display certain characters:

```
char a = 65, b = 66, c = 67;
cout << a;
cout << b;
cout << c;
```

```
ABC
```

Tip: A list of all ASCII values can be found in our ASCII Table Reference.

2.11. String Data Types

2.11.1 String Types

The `string` type is used to store a sequence of characters (text). This is not a built-in type, but it behaves like one in its most basic usage. String values must be surrounded by double quotes:

```
string greeting = "Hello";
cout << greeting;
```

To use strings, you must include an additional header file in the source code, the `<string>` library:

```
// Include the string library
#include <string>

// Create a string variable
string greeting = "Hello";

// Output string value
cout << greeting;
```

```
Hello
```

2.11.2 The auto Keyword

The `auto` keyword automatically detects the type of a variable based on the value you assign to it.

It helps you write cleaner code and avoid repeating types, especially for long or complex types.

For example: Instead of writing `int x = 5;`, you can write:

```
auto x = 5; // x is automatically treated as int
```

Starting in C++11, `auto` became a powerful way to let the compiler figure out the type based on the value you assign.

Here's an example showing how auto can be used to create variables of different types, based on the values you assign:

```cpp
#include <iostream>
#include <string>
using namespace std;

int main () {
  // Creating auto variables
  auto myNum = 5;             // int
  auto myFloatNum = 5.99;     // float
  auto myDoubleNum = 9.98;    // double
  auto myLetter = 'D';        // char
  auto myBoolean = true;      // bool
  auto myString = string("Hello"); // std::string

  // Print variable values
  cout << "int: " << myNum << "\n";
  cout << "float: " << myFloatNum << "\n";
  cout << "double: " << myDoubleNum << "\n";
  cout << "char: " << myLetter << "\n";
  cout << "bool: " << myBoolean << "\n";
  cout << "string: " << myString << "\n";

  return 0;
}
```

```
int: 5
float: 5.99
double: 9.98
char: D
bool: 1
string: Hello
```

- `auto` only works when you assign a value at the same time (You can't declare `auto x;` without assigning a value)
- Once the type is chosen, it stays the same. See example below:

```cpp
auto x = 5; // x is now an int
x = 10;     // OK - still an int
x = 9.99;   // Error - can't assign a double to an int
```

Note: In this tutorial, we usually use int, double, and other basic types when the type is simple and easy to see.

But for more complex types - like iterators and lambdas, which you will learn more about in a later chapter, we use auto to keep the code cleaner and easier to understand.

2.12. Data Types Examples

Here's a real-life example of using different data types, to calculate and output the total cost of a number of items:

2.12.1 Real-Life Examples

```cpp
#include <iostream>
using namespace std;

int main() {
  // Create variables of different data types
  int items = 50;
  double cost_per_item = 9.99;
  double total_cost = items * cost_per_item;
  char currency = '$';

  // Print variables
  cout << "Number of items: " << items << "\n";
  cout << "Cost per item: " << cost_per_item << currency << "\n
      ";
  cout << "Total cost = " << total_cost << currency << "\n";

  return 0;
}
```

```
Number of items: 50
Cost per item: 9.99$
Total cost = 499.5$
```

Chapter 3

Operators and Expressions

3.1. Operators

3.1.1 Arithmetic

3.1.2 Assignment

3.1.3 Comparison

3.1.4 Logical

3.1.5 Precedence

3.2. Strings

3.2.1 Concatenatio

3.2.2 Numbers and Strings

3.2.3 Strings Length

3.2.4 Access Strings

3.2.5 Special Characters

3.2.6 User Input Strings

3.2.7 Omitting Namespace

3.2.8 C-Style Strings

3.3. Math

3.3.1 Max and min

3.3.2 <cmath>Library

3.3.3 Complete Math Reference

3.4. Boolenas

3.4.1 Booleans

3.4.2 Boolean Expressions

3.4.3 Boolean Examples

Chapter 4

Control Structures

4.1. Decision Structures (Conditionals)

4.1.1 if

4.1.2 if...else

4.1.3 if..else if...else

4.1.4 switch

4.2. Repetition Structures (Loops)

4.2.1 for

4.2.2 while

4.2.3 do..while

4.3. Flow Control Within Loops

4.3.1 break

4.3.2 continue

4.3.3 return

4.4. Related Structures

4.4.1 Ternary Operator

4.4.2 goto

Chapter 5

Functions

Chapter 6

Funciones

Chapter 7

Punteros y Referencias

Chapter 8

Estructuras

Chapter 9

Manejo de Archivos de Datos

Chapter 10

Análisis Numérico

Chapter 11

Programación Orientada a Objetos

Chapter 12

Reference

iostream Library (Standard Input / Output Streams)

C++ iostream objects

The `<iostream>` library provides objects which can read user input and output data to the console or to a file.

A list of all iostream objects can be found in the table below.

Object	Description
cerr	An output stream for error messages
clog	An output stream to log program information
cin	An input stream that reads keyboard input from the console by default
cout	An output stream which writes output to the console by default
wcerr	The same as cerr but outputs wide char (wchar_t) data rather than char data
wclog	The same as clog but outputs wide char (wchar_t) data rather than char data
wcin	The same as cin but interprets each input character as a wide char (wchar_t)
wcout	The same as cout but outputs wide char (wchar_t) data rather than char data

HTML ASCII Reference

ASCII was the first character set (encoding standard) used between computers on the Internet.

Both ISO-8859-1 (default in HTML 4.01) and UTF-8 (default in HTML5), are built on ASCII.

The ASCII Character Set

- ASCII was the first character set (encoding standard) used between computers on the Internet.
- Both ISO-8859-1 (default in HTML 4.01) and UTF-8 (default in HTML5), are built on ASCII.
- ASCII stands for the “American Standard Code for Information Interchange”.
- It was designed in the early 60’s, as a standard character set for computers and electronic devices.
- ASCII is a 7-bit character set containing 128 characters.
- It contains the numbers from 0-9, the upper and lower case English letters from A to Z, and some special characters.
- The character sets used in modern computers, in HTML, and on the Internet, are all based on ASCII.
- The following tables list the 128 ASCII characters and their equivalent number.

ASCII Printable Characters

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
32	space	42	*	52	4	62	>	72	H
33	!	43	+	53	5	63	?	73	I
34	"	44	,	54	6	64	@	74	J
35	#	45	-	55	7	65	A	75	K
36	$	46	.	56	8	66	B	76	L
37	%	47	/	57	9	67	C	77	M
38	&	48	0	58	:	68	D	78	N
39	’	49	1	59	;	69	E	79	O
40	(	50	2	60	<	70	F	80	P
41	)	51	3	61	=	71	G	81	Q
82	R	92	\	102	f	112	p	122	z
83	S	93	]	103	g	113	q	123	{
84	T	94	^	104	h	114	r	124	|
85	U	95	_	105	i	115	s	125	}
86	V	96	`	106	j	116	t	126	~
87	W	97	a	107	k	117	u		
88	X	98	b	108	l	118	v		
89	Y	99	c	109	m	119	w		
90	Z	100	d	110	n	120	x		
91	[	101	e	111	o	121	y		

ASCII Device Control Characters

The ASCII control characters (range 00-31, plus 127) were designed to control hardware devices.

Control characters (except horizontal tab, line feed, and carriage return) have nothing to do inside an HTML document.

Dec	Abbr.	Description	Dec	Abbr.	Description
0	NUL	Null character	17	DC1	Device Control 1
1	SOH	Start of Header	18	DC2	Device Control 2
2	STX	Start of Text	19	DC3	Device Control 3
3	ETX	End of Text	20	DC4	Device Control 4
4	EOT	End of Transmission	21	NAK	Negative Acknowledge
5	ENQ	Enquiry	22	SYN	Synchronous Idle
6	ACK	Acknowledge	23	ETB	End of Trans. Block
7	BEL	Bell	24	CAN	Cancel
8	BS	Backspace	25	EM	End of Medium
9	HT	Horizontal Tab	26	SUB	Substitute
10	LF	Line Feed	27	ESC	Escape
11	VT	Vertical Tab	28	FS	File Separator
12	FF	Form Feed	29	GS	Group Separator
13	CR	Carriage Return	30	RS	Record Separator
14	SO	Shift Out	31	US	Unit Separator
15	SI	Shift In	127	DEL	Delete
16	DLE	Data Link Escape			