C++ Basics

A Language with Some Class

Mustafif Khan

Contents

1. Getting Started

C++ is a general pupose programming language, created at 1985 by Bjarne Stroustrup as at first an extension of the Grandpa language, C. C++ or also called `"C with Classes"`, was first developed for system and embedded programming, however over the years it has made itself seen in game development, desktop application development, and has overall been a very good language to have in your portfolio.

Little known fact, the first Command Line program written in MKProjects were first written in C++ before being written in Rust. For C++ there is a well known compilers you can use g++.

1.1 Installing G++

```
# Mac OS
$ g++

# Debain Linux
$ sudo apt-get install g++
```

Usage:

```
$ g++ file.cpp # to compile C++ program
$ ./a.out # to execute program binary
```

2. Introduction

2.1 Program Structure

Consider the following hello world program, `hello.cpp`:

```
#include<iostream>

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

The program runs from top to bottom, line by line:

- The first line instructs the compiler to locate the file that contains a library called `iostream`.
- This library contains code that allows for I/O (input & output).
- The `main()` function houses all the instructions for the program.

2.2 Basic Output

Now let's talk more about the `std::cout` in our program above. This is used to display output to the user's command line or terminal. To use `std::cout`, you must use it following << and a string or variable you wish to output.

```
std::cout << "The answer of the test is: " << answer << std::endl;
```

Note: `std::endl` is used to end the line of the output.

2.3 Comments

Comments are useful to document code, temporary debugging and in C++, it supports two different type of comments, single line `//` and multi-line `/* */`. Comments are ignored by the compiler at compiler time, making them a very good way to organize your code.

```
// This single line will be ignored

/*
The first C++ program written by MKProjects
was only available for Linux on Sanp!
All of this will be ignored !!!
*/
```

Compile & Run

Since we have our program, `hello.cpp`, we may as well compile and run it.

```
# First compile the program with g++
$ g++ hello.cpp

# Now run the binary to execute the program
$ ls
a.out hello.cpp

$ ./a.out
Hello World
```

3. Variables

A variable refers to a storage location in the computer's memory that one can set aside to save, retrieve, and manipulate data.

To initialize a variable in C++, you must first declare it's data type (will be discussed next), a name for the variable and assigned a value with the assignment operator " ="
Example: `int var = 10 // data_type name = value; \verb`

Variables can also be declared uninitialized, this means it doesn't have a value yet, and this is done by not adding an assignment value.

3.1 Data Types

In C++, there are 4 primitive data types, these are:

1. integers (`int`)
2. double floating point (`double`)
3. characters (`char`)
4. boolean (`bool`)

As well as a common data type used along the primitive type is strings(`std::string`).

3.1.1 Integers

The integer data type is a non-decimal number that can be positive or negative. An integer variable is declared with the `int` keyword, and keep in mind it can not be manipulated along the double data type. An integer typically requires 4 bytes of memory space and ranges from -2^{31} to 2^{31}.
Example: `int foo = 89`

3.1.2 Doubles

The double data type is a decimal point number requiring 8 bytes of memory, and is declared using the `double` keyword.
Example: `double foo = 0.78`

3.1.3 Characters

The character data type is a single character that is wrapped within single quotes `' '`. They typically require 1 byte of memory, and is declared by the `char` keyword.
Example: `char letter = 'A'`

3.1.4 Strings

Strings are an array of characters and are wrapped within double quotes `" "`. To declare a string you will need to use `std::string`.
Example: `std::string word = "hello";`

3.1.5 Boolean

A boolean data type is a value that is either `true` or `false`, and is declared using the `bool` keyword
Example: `bool condition = true`

3.2 Arithmetic Operators

C++ supports different types of arithmetic operators that can perform common mathematical operations:

- + addition
- - subtraction
- * multiplication
- / division
- % modulo (yields the remainder)

3.3 User Input

`std::cin` which stands for “character input”, reads user input from the keyboard. Here, the user can enter a number, press `enter`, and that number will get stored in `tip`.

```
int tip = 0;

std::cout << "Enter amount: ";
std::cin >> tip;
```

4. Conditionals & Logic

4.1 if Statement

An `if` statement is used to test an expression for truth.

If the condition evaluates to true, then the code within the block is executed; otherwise, it will be skipped.

```
if (a == 10) {
  // Code goes here
}
```

4.2 else Clause

An `else` clause can be added to an if statement.

- If the condition evaluates to true, code in the if part is executed.
- If the condition evaluates to false, code in the else part is executed.

```
if (year == 1991) {
  // This runs if it is true
}
else {
  // This runs if it is false
}
```

4.3 else if Statement

One or more `else if` statements can be added in between the if and else to provide additional condition(s) to check.

```
if (apple > 8) {
  // Some code here
}
else if (apple > 6) {
  // Some code here
}
else {
  // Some code here
}
```

4.4 switch Statement

A `switch` statement provides a means of checking an expression against various cases.

- If there is a match, the code within starts to execute.
- The `break` keyword can be used to terminate a case.
- `default` is executed when no case matches.

```
switch (grade) {
  case 9:
    std::cout << "Freshman\n";
    break;
  case 10:
    std::cout << "Sophomore\n";
    break;
  case 11:
    std::cout << "Junior\n";
    break;
  case 12:
    std::cout << "Senior\n";
    break;
  default:
    std::cout << "Invalid\n";
    break;
}
```

4.5 Relational Operators

Relational operators are used to compare two values and return true or false depending on the comparison:

- == equal to
- =! not equal to
- >greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

4.6 Logical Operators

Logical operators can be used to combine two different conditions.

- `&&` requires both to be true (and)
- `||` requires either to be true (or)
- ! negates the result (not)

5. Loops

5.1 While Loops

A `while` loop statement repeatedly executes the code block within as long as the condition is true. The moment the condition becomes false, the program will exit the loop.

Note that the while loop might not ever run. If the condition is false initially, the code block will be skipped.

```
while (password \verb!= 1234) {

  std::cout << "Try again: ";
  std::cin >> password;

}
```

5.2 For Loops

A `for` loop executes a code block a specific number of times. It has three parts:

- The initialization of a counter (`int i = 0`) - The continue condition (`i < 10`) - The increment/decrement of the counter (`i++`)

This example prints 0 to 9 on the screen.

```
for (int i = 0; i < 10; i++) {

  std::cout << i << "\n";

}
```

6. Vectors

In C++, a vector is a dynamic list of items, that can shrink and grow in size. It is created using `std::vector<type> name;` and it can only store values of the same type.

To use vectors, it is necessary to `#include` the vector library

```
#include <iostream>
#include <vector>

int main() {

  std::vector<int> grades(3);

  grades[0] = 90;
  grades[1] = 86;
  grades[2] = 98;

}
```

During the creation of a C++ vector, the data type of its elements must be specified. Once the vector is created, the type cannot be changed.

6.1 Vector Indexes

An index refers to an element's position within an ordered list, like a vector or an array. The first element has an index of `0`.

A specific element in a vector or an array can be accessed using its index, like `name[index]`.

```
std::vector<int> grades = {65, 78, 90, 85}

std::cout << grades[2];
// Outputs: 90
```

6.2 Vector Sizes

The `.size()` function can be used to return the number of elements in a vector, like `name.size()`.

```
std::vector<std::string> employees;

employees.push_back("michael");
employees.push_back("jim");
employees.push_back("pam");
employees.push_back("dwight");

std::cout << employees.size();
// Prints: 4
```

6.3 Push and Pop

The following functions can be used to add and remove an element in a vector:

- `.push_back()` to add an element to the "end" of a vector
- `.pop_back()` to remove an element from the "end" of a vector

```
std::vector<std::string> wishlist;

wishlist.push_back("GTX 3090");
wishlist.push_back("Ryzen 5 5600");

wishlist.pop_back();

std::cout << wishlist.size();
// Prints: 1
```

7. Functions

A **function** is a set of statements that are executed together when the function is called. Every function has a name, which is used to call the respective function.

A C++ function has two parts:

- Function declaration
- Function definition

The declaration includes the function's name, return type, and any parameters.

The definition is the actual body of the function which executes when a function is called. The body of a function is typically enclosed in curly braces.
A function can look like the example below:

```
#include<iostream>
void hello(); //Function Declaration

int main(){
    hello(); //Function call
    // Prints Hello!
}
void hello(){ //Function Definition
    std::cout << "Hello!";
}
```

7.1 Parameters

Function parameters are placeholders for values passed to the function. They act as variables inside a function. They are placed in the parantheses in the function declaration, `int add(int a, int b)`.

Consider the example below:

```
#include<iostream>
void print_int(int i);
```

```
int main(){
    print_int(10); //Passes the value of 10 to the parameter i
    //Prints 10
}
void print_int(int i){
    std::cout << i;
}
```

7.2 Return Values

A function that returns a value must have a `return` statement. The data type of the return value also must match the method's declared return type.

On the other hand, a `void` function (one that does not return anything) does not require a return statement.

Consider the example below:

```
int add(int a, int b);

int main(){
   int sum = add(6, 8); // sum = 14

}
void add(int a, int b){
    return a+b;
}
```

7.3 Scope

The scope is the region of code that can access or view a given element:

- Variables defined in global scope are accessible throughout the program.
- Variables defined in a function have local scope and are only accessible inside the function.

```
#include <iostream>

void print();

int i = 10;       // global variable

int main() {
  std::cout << i << "\n";
}
void print() {
  int j = 0;      // local variable
  i = 20;
  std::cout << i << "\n";
  std::cout << j << "\n";
}
```

8. Classes and Objects

A C++ class is an user-defined data type that may contain it's own unique methods, attributes, etc. To define your own class, use the `class` keyword along with a useful name for it.

We will be defining our own class, and use it to describe the next sections:

```
#include<strings>
class Summon{
    // Class attributes
    std::string name;
    char type;
    int tier;
    std::string description;

    // Constructor
    Summon(std::string name, char type, int tier, std::string description);

    //Private Methods:
    private:
    int what_dmg(Summon s){ //Finds dmg of a perticular summon
        if (s.type == 'T'){
            return (s.tier * 2 )-1;
        } else (if s.type == 'S') {
            return s.tier * 2;
        } else {
            return 0;
        }
    }
    std::string what_type(Summon s){
        if (s.type == 'T'){
            return "Tech";
        } else if (s.type == 'S'){
            return "Striker";
        }

    }
    //Public Methods:
    public:
    void info(Summon s){
        std::cout  << "Name: "<< s.name << "\n"
        << "Type: " << what_type(s) << "\n"
        << "Dmg: " << what_dmg(s) << "\n"
        << "Description: " << s.description << std::endl;
    }
};
```

8.1 Class Members

A class is comprised of class members:

- Attributes, also known as member data, consist of information about an instance of the class.
- Methods, also known as member functions, are functions that can be used with an instance of the class.

This can be seen in our program above, things like `char type;` or `int tier;` are all class attributes. Our class methods were split into `private` and `public` methods *(talked in Access Control below)*, and are such things like `what_dmg` or `info`.

8.1.1 Objects

In C++, an object is an instance of a class that encapsulates data and functionality pertaining to that data. This can be in our functions like `what_type(Summon s)` that uses the object `Summon s`.

8.2 Constructor

For a C++ class, a constructor is a special kind of method that enables control regarding how the objects of a class should be created. Different class constructors can be specified for the same class, but each constructor signature must be unique.

We have a constructor in our class, and is seen as `Summon(std::string name, char type, -int tier, std::string description)`, and what it means is that to initialize a Summon instance, you must have the following.

8.3 Access Control

C++ classes have access control operators that designate the scope of class members:

- `public`
- `private`

`public` members are accessible everywhere; `private` members can only be accessed from within the same instance of the class or from friends classes.

You can see this when we don't want people to use our `what_dmg` or `what_type` functions, so we can avoid that by making those private, while info will be public, and use these private functions.

9. References and Pointers

9.1 Memory Address

In C++, the memory address is the location in the memory of an object. It can be accessed with the "address of" operator, `&`.

Given a variable `random_var` the memory address can be retrieved by printing out `&random_var`. It will return something like: `0x7ffd7caa5b54`.

9.2 Pointers

In C++, a pointer variable stores the memory address of something else. It is created using the `*` sign.
Example: `int* pointer = &gum; //Gets address of something like 0x3fed7c9a8b578`

9.3 References

In C++, a reference variable is an alias for another object. It is created using the `&` sign. Two things to note:

1. Anything done to the reference also happens to the original.
2. Aliases cannot be changed to alias something else.

Example: `int &a = b; //Now a shares the same address as b`

9.4 Pass-By-Reference

In C++, pass-by-reference refers to passing parameters to a function by using references.

It allows the ability to:

- Modify the value of the function arguments.
- Avoid making copies of a variable/object for performance reasons.

```
void swap_num(int &i, int &j) {
  int temp = i;
  i = j;
  j = temp;
}

int main() {
  int a = 100;
  int b = 200;

  swap_num(a, b);

  std::cout << "A is " << a << "\n"; //Prints 200
  std::cout << "B is " << b << "\n"; //Prints 100
}
```

9.5 const Reference

In C++, pass-by-reference with `const` can be used for a function where the parameter(s) won't change inside the function.

This saves the computational cost of making a copy of the argument.

```
int triple(int const &i) {

  return i * 3;

}
```

9.6 Dereference

In C++, a dereference reference operator, `*`, can be used to obtain the value pointed to by a pointer variable.

```
int gum = 3;

// * on left side is a pointer
int* pointer = &gum;

// * on right side is a dereference of that pointer
int dereference = *pointer;
```