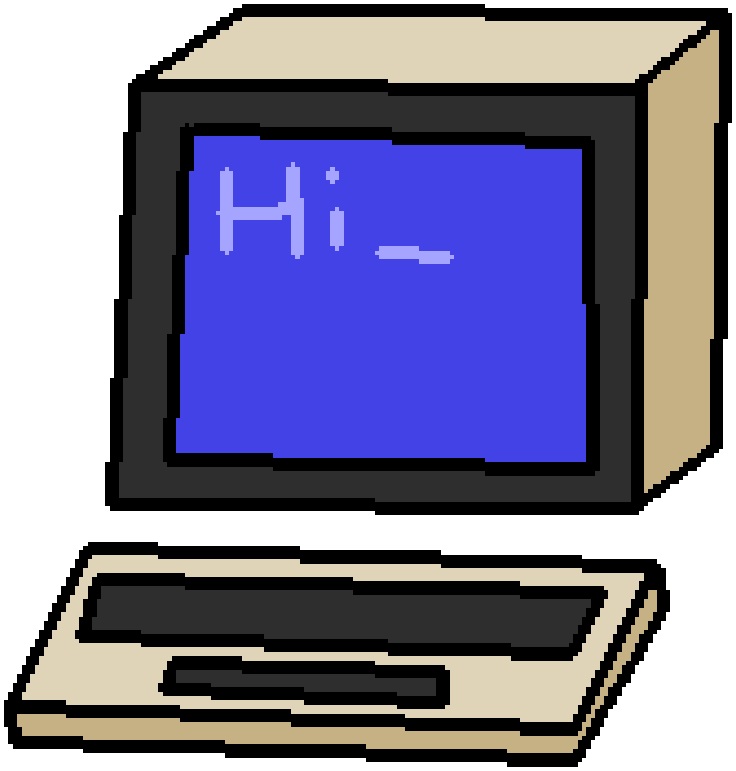
Core Computer Science Notes: Introduction/C++ Basics



An overview compiled by Rachel Singh

Last updated June 8, 2020

Contents

Topic 1

Basic C++ Programs

In C++, every program begins at the `main()` **function**.

```
1 int main()
2 {
3     // Program code goes here
4
5     return 0;
6 }
```

We will eventually be writing our own functions, but to start out with we will be putting all of our program logic within `main`, after the opening curly-brace { and before the `return 0;`.

int main(): "main" is the name of the function, and every C++ program begins at the main function. If you named this something else, C++ would complain because it wouldn't know where to start! The main has a return type "int", which means that the program expects some integer to be returned at the end.

{ } : The opening and closing curly braces in C++ (and similar languages like Java, C#, and JavaScript) denote the beginning and ending of a **code-block** - some code that belongs together.

return 0;: This returns a "normal status code" of 0. In older software, we'd use number codes to represent different errors, with 0 being "no errors". Since main has an "int" return type, we must return 0 at the end.

This might not make perfect sense to you right off the bat. Sometimes, you

need to take certain things for granted, and you will learn about them more in-depth as you learn more about C++.

1.1 Comments

In programming, it is often useful to add **comments** to your code. The computer doesn't use these comments at all, but it is useful for other programmers (or for yourself in the future) to be able to see comments to understand what's going on.

Single-line comments: You can write a single-line comment like this:

```
1 // The following gets the distance between 2 points
```

Multi-line comments: You can also add comments on multiple lines like this:

```
1 /*
2 My program
3 By me
4 This program calculates distances
5 */
```

1.2 Variables

In a basic C++ program, you will usually at least have **variables** in order to store data. If you're not storing data and processing that data, then what's the point of writing a program?

Declaring a variable: In C++, when you want to start using a new variable, you must **declare** it first. At minimum, you need to specify the variable's **data type** (what kind of data it stores) and the variable's **name or identifier** (what you call the variable).

```
1 int candiesPerKid;
```

The first item (int) is the **data type** and the second item (candiesPerKid) is the **variable name**.

Variable names: Variable names in C++ can consist of **letters (upper and lower case)**, **numbers** (except it cannot start with a number), and **underscores**. Other symbol types cannot be used, and certain words cannot be used if they're special keywords in C++ (for example, "if" is a keyword, so it can't be a variable name.)

In C++, variable names are generally written in camelcase, with the first letter staying lower-case:

```
1 int howManyDays;
2 float howMuchMoney;
3 string studentName;
```

Initializing a variable: The first time you assign a value to a variable it is known as **initializing** it. You can declare a variable and initialize it later on...

```
1 int candiesPerKid;
2 // stuff happens
3 candiesPerKid = 2;
```

Or you can declare a variable and initialize it at the same time...

```
1 int candiesPerKid = 2;
```

Variables and memory: When a variable is declared, it's going to take up a little space in RAM - in memory. When variables aren't used anymore, they will be freed for other programs to use.

Garbage values: C++ doesn't automatically initialize your variables with information. If you declare a variable and *never initialize it*, there is no way to know what value that variable has. In other words, it will store "garbage": data it pulls from memory, any sort of "old data" that was used by a previous program or variable that has since stopped using that address.

1.2.1 Named constants

In many cases, we will want to use **named constants** that look like and behave like variables, but whose value cannot be changed after declaration. This is useful when we're going to use a number a lot of times (e.g., "maxStudents = 30") but we don't want to hard-code the number 30 all through

our program. Named constants are declared like variables, but use the `const` keyword at the beginning, and must be initialized with a value right away:

```
1 const int MAX_SEATS_IN_CAR = 5;
2 const string CURRENCY = "USD";
```

This also helps in that, if we want to change the number later on, we only have to update it in one location - where the const is being declared.

Name constants usually are named with ALL_UPPER_CASE_LETTERS, with underscores between each word.

Example:

```
1 const int CANDIES_PER_KID = 2;
2
3 int kids;
4 cout << "How many kids? ";
5 cin >> kids;
6
7 int totalCandies = kids * CANDIES_PER_KID;
8
9 cout << "Please order " << totalCandies << " candies."
     << endl;
```

1.2.2 Data types

When declaring a variable, we must specify what **data type** it is: what kind of data it will store. In some languages like Python, a variable can store integers one minute then strings the next - this is not possible in C++. Once a variable is declared, it can only store data of the same type.

Data type	Size	Description

Booleans	1 byte	Stores `true` or `false`.
		`bool savedGame = true;`
		`bool programDone = false;`

Characters	1 byte	Stores a single character. Char literals must be in single quotes.
		`char currency = '$';`
		`char answer='y';`

Integers	2 bytes	Stores whole numbers (positive/negative/zero).
		`int totalStudents = 10;`
		`int year = 1980;`

Floats	4 bytes	Stores numbers with a decimal component.
		`float price = 9.99;`
		`float ratio = 0.50;`

Doubles	8 bytes	Same as a `float`, but double the precision.

Strings	(variable)	Stores any kind of data. String literals must be within double quotes.
		`string name = "JCCC";`
		`string area = "(913)";`

1.3 Input and Output

Some programs don't interact with the user at all. These are usually called **scripts** and do a series of commands to speed up multi-step processes. However, our programs will often have a **user interface** that the user can interact with. We will output information to the screen and get input from the user.

To be able to do input and output in C++, we need to make sure to include the **iostream** library at the top of our program.

```
1 #include <iostream>      // cout and cin
2 using namespace std;
3
4 int main()
5 {
6     // Program code goes here
7
8     return 0;
9 }
```

1.3.1 Outputting data

We use the `cout` command to output data to the screen. We also use `<<`, the **output-stream operator** to link **string literals** and **variables** in our output statement.

Outputting a string literal: A **string literal** is a hard-coded string that you write in your program. String literals must be written in double-quotes, and will be displayed as-is.

Code:

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

Output:

```
Hello!
```

Outputting a variable value: If you use `cout` and a variable's name, it will display the value stored in the variable:

Code:

```
1 int cakes = 10;
2 cout << cakes << endl;
```

Output:

```
10
```

Often, just outputting the variable on its own isn't enough data for the user. You will usually need to add a string literal to write out a label for what is being shown:

Code:

```
1 int cakes = 10;
2 cout << "Total cakes: " << cakes << endl;
```

Output:

```
Total cakes: 10
```

New lines: C++ won't automatically put line breaks in your program output. If you did several `cout` statements like this, the output would be all on the same line:

Code:

```
1 string name1 = "Microcenter";
2 string name2 = "TableTop";
3 string street = "Metcalf";
4
5 cout << name1;
6 cout << name2;
7 cout << street;
```

Output:

```
MicrocenterTableTopMetcalf
```

You can manually add new lines by linking `endl` into your stream, or you can add the string literal `"\n"`.

Code:

```
string name1 = "Microcenter";
string name2 = "TableTop";
string street = "Metcalf";

cout << name1 << endl;
cout << name2 << "\n";
cout << street;
```

Output:

```
Microcenter
TableTop
Metcalf
```

Special characters: You can add special characters in your string literal to do different things:

Character	What it does
`"\n"`	New line
`"\t"`	Add a tab
`"\\"`	Display a backslash
`"\" "`	Write a double-quote
`"\a"`	Make a bell sound

1.3.2 Getting input

One way to get input from the user is to use the `cin` command and the **input-stream operator** `>>`. This is good for getting integers, floats, doubles, and one-word strings. Any time we get input from the user, we must store their input in a variable somewhere, so make sure you declare a variable before using `cin`.

```
1  string username;
2  int age;
3  float money;
4
5  cout << "Enter your username: ";
6  cin >> name;
7
8  cout << "Enter your age: ";
9  cin >> age;
10
11 cout << "Enter your money: ";
12 cin >> money;
```

Besides needing a variable declared before each `cin` statement, you should usually also display output with `cout` to let the user know that you're expecting some information from them.

Getting a line of text In some cases, you will want to get a `string` from the user that may include spaces. Using `cin >>` won't work on strings with spaces, so we have to use a special function: `getline`.

```
1  string fullname;
2  cout << "Enter your full name then press ENTER: ";
3  getline( cin, fullname );
```

Using `getline` requires open and closing parentheses (because it's a function), and two pieces of data: `cin` and the variable to store the data in.

Program errors: If you inter-mix `cin >> ...` and `getline()`, you will probably run into **buffer errors** in your program - often, certain inputs will be skipped. When you're using both of these command types...

Always use `cin.ignore();`
before a `getline()` that was preceeded by a `cin >>`!