C++ for Beginners - CyberU

# First Program

**Note:** The course uses Codeblocks. I will use VS 2019 Community unless something cannot be completed otherwise.

Mingw comes with Codeblocks installation. Ensure environment variable is installed pointing to /bin of Codeblocks Mingw before attempting to run executable in CLI.

## Build Configuration / Targets

Codeblocks > Properties > Build Targets

We add a define for Release Target through #defines called ExtraPlanet. Now we can go back to our code and write what is know as a **compiler directive** to check for this define.

#ifdef ExtraPlanet // if this define is found, increment planetCount.

planetCount++;

#endif

## Variables

int – a data type

number – name or identifier for the variable to give it meaning

= - assignment operator, assign whatever is on the right hand side to left hand side

5 – a literal

**Declare variables to ensure that variables are not given random values among other things**.

### Naming Conventions

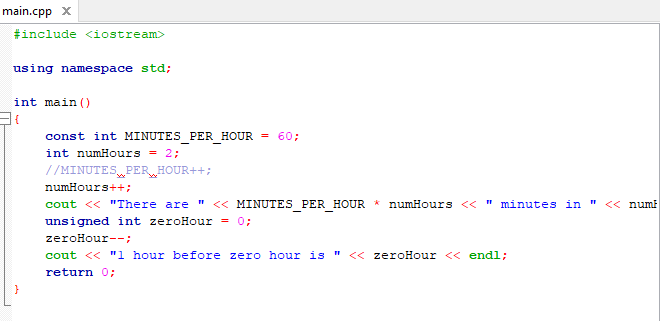
Hungarian Notation – when you start the variable name with the first letter of its data type.

int iminutes = 0;

string sname = “”;

In programming languages, it is conventional to name **constant** variables using uppercase and underscore for word separation.

**Labs 01**



Notice that for an unsigned int decrementing will go back to the max value which is 4294967295. This allows us to have a nonnegative number stored in memory, ranging from 0 to 4294967295.

iomanip ([C++ Library - <iomanip> - Tutorialspoint](https://www.tutorialspoint.com/cpp_standard_library/iomanip.htm)) is a library that is used to manipulate the output of a C++ program.

### Data Types

**Bool** – primitive data type yes no 1 or 0. Any non-zero value is considered true, whereas 0 is considered false. Bool test = 55 represents a true state for test variable. Even -5 will be true.

**Char** – a data type that stores a character. It can be a letter symbol or even a number. It is important to enclose the literal character in single quotes otherwise if double quotes are used it will be interpreted as a string. ex. char c = ‘C’;

Given that char holds numerical variable we can do operations on the literal character value. For instance, if we increment the above statement by 1, we get D. If we add 32 to an uppercase character, we get the lowercase equivalent.

If let’s say we add 256 to the character, the line is skipped as adding 256 to a byte does nothing to it. If on the other hand we add 257, it will roll over like modulus and increment our char to represent D. it’s like you only really added one, as it rolls around.

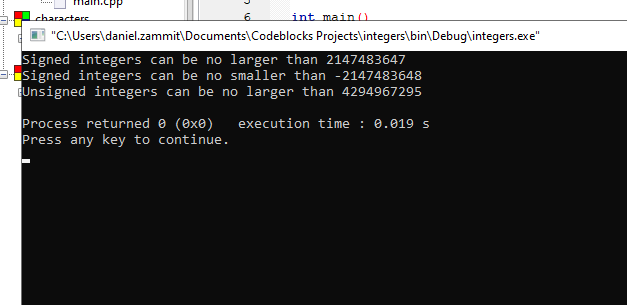
**int** – a representation of a whole number. Cannot hold anything with a point/fraction. Integer can be signed, which means they can positive or negative, or unsigned which means only positive.

If we try to assign an unsigned int variable with a negative value, it will be assigned the max value for the unsigned int taking away the attempted assigned negative value.

ex. unsigned int test = -3;

test is equal to 4,294,967,295 which is the max number for an unsigned 32-bit integer take away -3 so it will actually equal to 4294967293 as any value below 0 wraps around to max value.

**climits** is a library used to help us with limits of variables such as integers. This gives us an idea using INT\_MAX / INT\_MIN of the corresponding limits for each type.



## Error Handling and Data Types

In console application typically messages are sent to the standard output stream. There is also another stream known as the error stream, which means console applications handling errors can have different behaviours depending on the type of error. ex. display message, save file on error.

Instead of outputting to cout we output to **cerr**. We can also build logic to output error from the stream followed by a non-zero return code to symbolise different types of errors should this be the case.

### Long and Short Data Types

We can use the **sizeof** method to get the number of bytes that are used to store our variable. Note this is not the actual value of the variable but how much memory the variable type consumes.

For example in the case of an *unsigned int* the number of bytes to store in memory is 4. The reason why we care about the size of our variable is that the CPU takes a certain amount of time to process a byte of information. The more bytes, the larger the variable, the slower the CPU. Scaling up to a million times a variable, it could take 2mins or an hour. The decision would be choosing between maximum value it can hold vs number of bytes.

**Long** data type comes in hand when we need to store larger numbers.

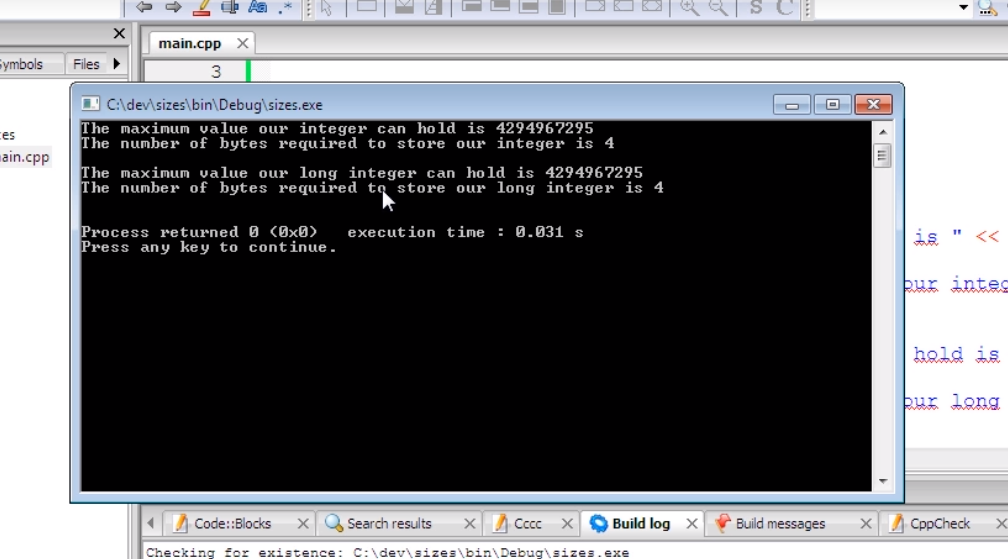


Figure 1 - Compiler and CPU architecture will dictate the size of variables, it also depends on platform

In this case above, a long integer is a synonym for an integer, but again this depends on architecture, compiler and platform.

If we want to store something even larger, there is the **long long** integer data type. It is always the struggle to find the balance between storage and performance.

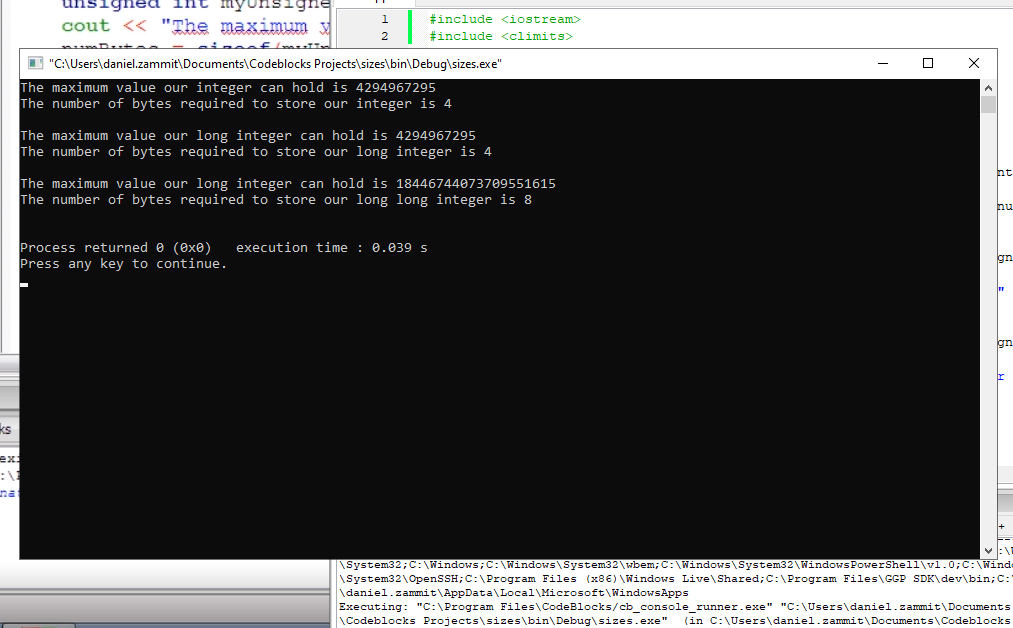


Figure 2 - Long Long is double the size of long

For more performance when we do not require a large variable to store smaller value numbers, we can use the **short** modifier. This has a raw value max of 65565 as it is half a normal int. Then again depends on compiler and platform. It is faster and more memory efficient to use the short modifier when we can.

### Char Single Byte

As stated above a char needs a single byte of memory for storage. Now when this is not enough, let’s say we have an alphabet that contains several types of characters, like Chinese or Japanese, in this case we can use a wide char, or **wchar\_t**. This is how it is declared in C++ and we do not write any modifiers with it as it can have unpredictable behaviour, according to the CPP specification.

**NOTE:** floats also have different sizes, but they will be covered later.

### Casting

Converting between datatypes.

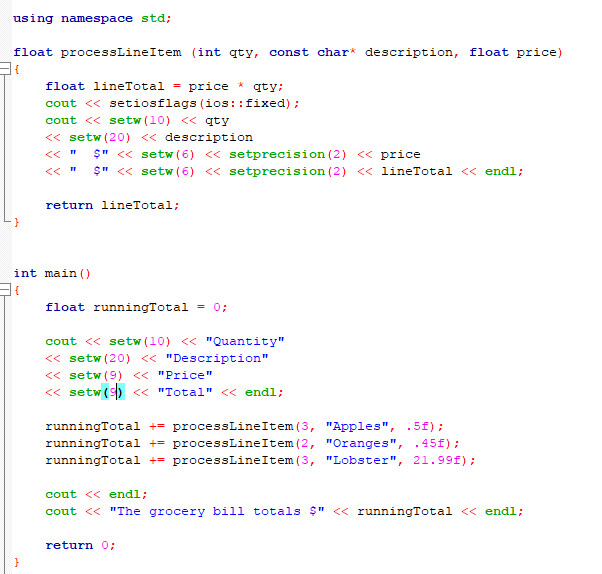
int myConvertedChar = (int)myChar;

The operator in parentheses means cast this datatype (myChar) to type **int**, to store it to LHS and initialise myConvertedChar to the casted value of myChar.

bool and char casting works to int and no data is lost. In the case of floating numbers, remember they are inaccurate and can lose accuracy of data. In many cases you will lose data so you need to be careful when to cast and how.

### Formatting Output

In order to format output we have a good helper class known as ***iomanip***. We can call a method *setw(int colomnwidth)* taking a column width value as a parameter before outputting a string. by default *setprecision*(value) will cause a float to display a minimum number of the supplied digits / minimum of the digits value supplied in its parameter. So we set ios::fixed flag to tell iostring to use fixed precision instead of number of digits after the decimal. With the fixed flag, even if a float is only one digit, it will be rounded up to the 2-digit minimum. ex. 0.5f to 0.50 on output.



### Input from STDIN

cin >> inputA; Cin is smart enough to do any casting / type conversion. It knows anything coming from keyboard is char or string and it is smart enough to convert to int, should this be the case.

**Note**: always make sure you validate input correctly to avoid any logical programming errors, such as confusing cin with characters when it is expecting numbers. Also, cin by default only processes up to the end of white space or a new line. If you type full name in one go, chances are it will only take first name as input as it will stop processing after the space.

A way to get around this white space limitation for cin is using a helper function known as **getline**.

## Operators and Precedence

**Comments**

It is very important to write useful comments instead of just translating written code into normal sentences. People can read code so we need to have more useful comments than just translations. A comment should describe what it is we are doing for that line or section, what the objective is, think about the bigger picture and don’t just repeat what the code is doing.

// single line comment

/\*\*/ block comment

**Meaningful variable names vs cryptic**

Do not be ambiguous with naming, be specific and don’t use anything that might use syntax keywords. Use meaningful names for parameters to make it easier to read and understand code later. Avoid abbreviations unless they are common terms. This also applies for function name, leading to understandable and more readable code.

### Assignments and Operators

**=** assignment operator: take what we have on the RHS and store it to the variable on our LHS.

Float myPrice = 200.0f; Assign the variable myPrice the Literal float value of 200;

<< insertion operator: associated with iostream library objects, meaning take value on RHS pass it to the object and return the reference to that object.

Cout << “Hello : ” << myName << “ ” << mySurname;

We pass my surname to “”, then “” to myName then to Hello until it reaches cout, so it keeps on getting a reference to the previous object, meaning we can chain objects for output on the cl stream.

\* + - arithmetic operators taking both operands on each side and working on them to produce an outcome to assign the LHS.

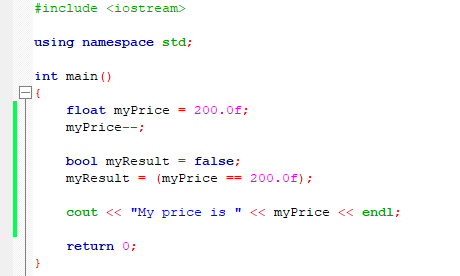
% modulus operator: LHS divided by RHS but take remainder as the result

+= compound assignment: shortcut to saying add to the current value and reassign the new value back to the same variable

++ increment operator: add 1 and store back to the same variable. Same goes for decrement operator.

There are other types of operators such as ternary and bitwise operator, which will be handled in another topic.

### Comparators and Logical Operators

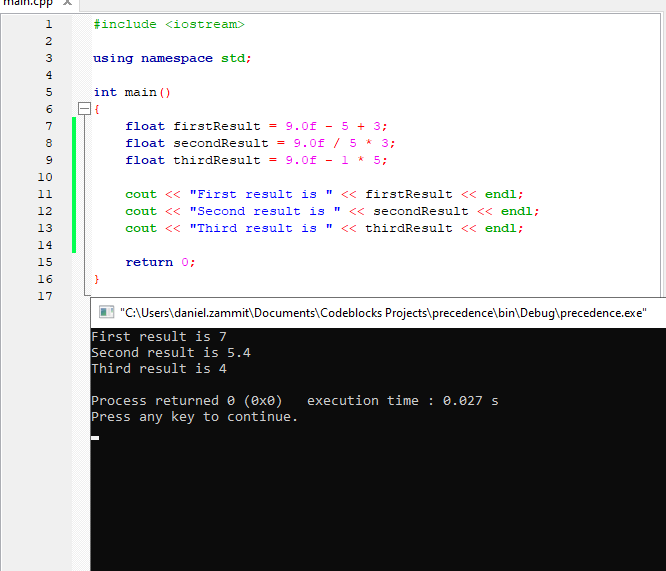


Important to use comparison expression here. If by mistake we remove an equal sign, the whole expression would turn into an assignment of 200 to variable myPrice and give us an unexpected result.

! NOT operator : inverse logic of the current value of variable or expression. LHS not RHS. Boolean opposite.

### Precedence

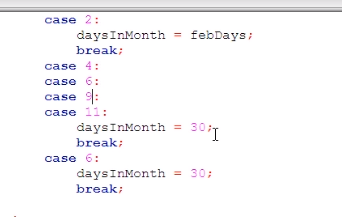
It’s all about BIDMAS, order of arithmetic execution. If we do not enclose operations within parenthesis, C++ will use it’s own order to calculate the arithmetic operations. We can see in this example that C++ uses BIDMAS to work out the third result, multiplying first before subtracting. Parenthesis always take precedence. There are some exceptions for this, however it will be discussed later.



## Decision Making and Program Flow

If / else / switch statements

In similar cases, we can use the break statement to our advantage, where we do not stop the execution flow if the cases can be handled in the same way. Example :



### Ternaries

Ternary refers to an expression that allows use to make logical decisions in a concise and readable way. A ternary statement return a value based on a comparison.

ex. daysInMonth = (year % 4 == 0) ? 29 : 28;

**do/while loop**

The usual do while and break statement to exit early if need be.

**For loop**

A way of repeating code within a program in a very controlled way. Continue command jumps to next line of execution and break command exits from the loop earlier, ideally used in an if statement.

## Arrays

Array is a set of variables of the same type. These are known as elements, residing side by side in memory and can be referenced using an index.

Shortcut: char monthName[] = “April”;

The compiler knows we are putting the string literal in a char array so it will break it up into individual char elements and add the “\0” null terminator at the end.

Be careful when trying to assign an array element which is outside the initialised scope as this will have undesired affects in our program.

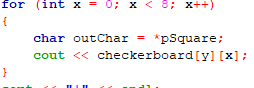
### Multi-dimensional arrays

An array of arrays, therefore in initialisation a proper notation is to use curly brackets to denote each array.

Check out checkers.cbp for nested loops.

### Pointers

Dereference and give us the actual value of what this pointer is pointing to.



## Binary Operations

**Binary Theory**

Representing binary in decimal value can be done by going from left to right, starting with the most significant bit if the binary string given is 8 bits longs, which means we know the value of the bits is constant.

Now a quick reminder on representing negative binary values. This can be done using Two’s complement.

To do this, basically we take the left hand most significant bit and turn it to 1, which will represent a negative value, essentially leaving the rest of the value to be positive and not larger than the intrinsic negative value.

**Bitset Helper function**

Instead of having own implementation to convert from binary to decimal and vice verse, we can use the bitset helper function to do this for us as shown below:

