C++ Basics - Review

This review will go through the basic steps to get you started in creating a C++ program.

Note: Any text in blue, like this text, contains optional information. While it is not required for this course, it contains useful information to assist you in forming better programming habits.

Topics

- *Inputs and Outputs*
- *Fundamental Data Types*
- *Assigning Variables*
- *Simple Operations*
- *Increment Operator*
- *Display Precision*

Here is some basic starter code: (This code is copyable!!)

```
#include <iostream>     // 1.)
using namespace std;    // 2.)

int main()              // 3.)
{

    return 0;           // 4.)
}
```

1) #include <iostream> - Includes the contents of <iostream> into our program.

a) The **#include** directive is a way of including a standard or user-defined files in your program, written at the beginning of your C++ program.

b) Including using angular brackets **(< >)** uses a predetermined directory path to access the file, usually used for accessing system **header files** located in your standard system directory.

c) **<iostream>** is one of many useful headers that defines the standard input/output stream objects:

std::cin	Standard input stream object
std::cout	Standard output stream object

i) The **std::** prefix tells the compiler that we are looking for an identifier inside the namespace (std), so the standard library won't conflict with any objects or functions named cout in the global namespace.

2) **using namespace std;**

   a) For our purposes, **using namespace std;** will be used, as we can reduce the necessity to use the prefix **std::** when using **cout** and **cin**, which are defined in the std namespace.

   b) On a side note, many programmers prefer to use the prefix *std::* to avoid conflicts with the global namespace, or user-defined namespaces; however, **this will not be required in this course.**

If you are interested in more, for a list of standard library headers go to the link below: All the functionality in the standard library listed in the link below is defined in the namespace std.

https://en.cppreference.com/w/cpp/header

---

3) **int main() {** *body* **}**

   a) Every program will include this line, which consists of 4 basic elements.

      i) A return type of **int**, or basic integer. This indicated the function will data of type **int.**

      ii) A global function called **main,** which will be used in every program that you create.

      iii) An optional argument list enclosed in parentheses ( ). For the purposes of this course, we won't be using arguments inside the main function; however, the parentheses ( ) are required.

      iv) The body enclosed in brackets { }, which will include our code. I encourage you to look at other programmers' code online, and develop your own indentation style, which you should keep consistent throughout your programs.

4) **return 0;**

   a) This line is needed because the main function, that has a return type of int, must return a value of type int.

   b) *return 0* is used to indicate a program ran successfully, so we will include this in the body of every main function that we create in this course.

Coding example

```cpp
/*****************************************************************//**
 * \file   myprogram.cpp
 * \brief  This program will prompt users for an integer, and
 *          display the results.
 *
 * \author Ryan Laur
 * \date   December 2020
 *********************************************************************/


#include <iostream>     // 1.)
using namespace std;    // 2.)

int main()              // 3.)
{
    cout << "Enter an integer : "; // prompting user
    int myInt = 0; // declare variable
    cin >> myInt; // storing input into var
    cout << "You entered : " << myInt << endl; // output var

    return 0;           // 4.)
}
```

- *Note, the comment section in the beginning is a useful documentation technique to make your code easier to understand. It usually includes the Author, date, filename, and a brief description of your program.*

- *Here, I used /file, /brief, etc. to document my code. This is a doxygen-style formatting that is widely used by programmers to generate documentation. In particular, I used a Visual Studio Extension to auto-generate this text.*

More information on this can be found here:

https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen

https://www.doxygen.nl/index.html

Variable Assignments

- **Identifier** – The name of a variable or anything else you must define is called an **identifier.**
    - Identifiers must start with a letter, and cannot contain symbols.
        - For example, *x123* or *_x12* are legal identifiers, but *123x* or *data-1* are illegal.

- **Variables** – Every variable in C++ must be *declared*, that is, you must tell the compiler what kind of data you will be storing into the variable.

Example:

```
int userNum;
double totalCost = 0;
```

1. The first line defines a variable with identifier, or name, **userNum** with data type **int** which means the variable can hold basic integers.

2. The second line defines a variable called **totalCost** with datatype **double**, and initializes it to zero.

Datatypes

```
int main()
{
    int x = 2;
    double y = 3.99;
    y = x; // this works as intended
    cout << y << endl;

    y = 3.2;
    x = y; // Avoid storing double into int!!!
    cout << x << endl;

    return 0;
}
```

- If you test this code, you will notice that the first output will be 2, as the variable y of type double can hold an integer without data loss.

- However, the second output will be 3, because if you store a double type into int, the compiler will automatically round down to the next whole integer, resulting in loss of information.

Important Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	modulo
+= -+ *= /=	Compound Assignment – This modifies the current value of a variable by performing an operation on it. x += 5; is the same as x = x + 5;

++
--
	Pre-increment and Post-increment
	1. ++x increments x by 1, and THEN evaluates the variable.
	2. x++ first evaluates the variable, and THEN increments x by 1.
	
	Example 1:
	
	x = 2;
	y = ++x;
	// x contains 3, y contains 3
	
	Example 2:
	
	x = 2;
	y = x++;
	// x contains 3, y contains 2

==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
&&	and
||	or

Setting Precision

```
double x = 2.799;
cout << fixed << showpoint;
cout.precision(2);
cout << x;
```

1. **cout << fixed –** This statement sets the floatfield format flag for the stream **cout**, so the floating-point values are written using fixed-point notation. Fixed is used in tandem with precision, as when you specify the precision, it only counts after the decimal place, showing the exact number of digits specified.

2. **cout << showpoint –** This statement sets the **showpoint** format flag for the stream **cout,** so the decimal point is always written for floating point values inserted into the stream, even for those whose decimal part is 0.

3. **cout.precision() –** This statement sets the maximum number of meaningful digits to display. If used with fixed, this number is the number of digits after the decimal space, indicating the exact number of digits after the decimal place to display.

Other useful operators:

::	Scope
. ->	Member access
<<	Bitwise shift left
>>	Bitwise shift right
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise exclusive OR
~	Unary complement (bit inversion) (NOT)
?:	Conditional operator – This operator evaluates an expression, returning one value if the expression is true, and another value if the expression evaluates to false. Syntax: • `condition ? result1 : result2` Example: a > b ? a : b This evaluates to a if a > b, and evaluates to b if a is not greater than b.