Chapter 2: Data Types

Before we can create variable to store our data, we need to learn what a data type is and how they work. There are a variety of built in data types that we can use immediately, but it is also possible to create your own.

This table details the data types that are found in C++ and you will be using throughout your studies.

|  |  |
| --- | --- |
| **Type** | **Keyword** |
| Boolean | bool |
| Character | char |
| Integer | int |
| Floating point | float |
| Double floating point | double |
| Valueless | void |
| Wide character | wchar\_t |

Table 2.1: Data types

The keyword is how this type is written in C++. If you make a mistake typing this in, for example miss a letter or use an uppercase character where it should be lowercase you will be alerted to a syntax error.

Not: Syntax is the term used for how the compiler expects to receive the code. If something does not match, you will receive a syntax error.

**Data Type Modifiers**

Each type reserves a certain amount of memory from the computer. In some situations you may find that you require the memory to be increased / decreased or you would like the value stored in a variable to be in a different range. To enable this there are data type modifiers.

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| signed | Use values in the negative and positive range. |
| unsigned | Use only the positive range. |
| short | Decrease the range, thus reducing the memory used. |
| long | Increase the range, which increases the memory used. |

Table 2.2: Data type modifiers

The following table details the data type, the amount of memory used and the range of values that can be stored in a variable of this type.

|  |  |  |
| --- | --- | --- |
| **Type** | **Typical Bit Width** | **Typical Range** |
| char | 1 byte | -127 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -127 to 127 |
| int | 4 bytes | -2,147,483,647 to 2,147,483,647 |
| usigned int | 4 bytes | 0 to 4,294,967,295 |
| signed int | 4 bytes | -2,147,483,647 to 2,147,483,647 |
| short int | 2 bytes | -32,768 to 32,768 |
| unsigned short int | range | 0 to 65535 |
| signed short int | range | -32,768 to 32,768 |
| long int | 4 bytes | -2,147,483,647 to 2,147,483,647 |
| unsigned long int | 4 bytes | 0 to 4,294,967,295 |
| signed long int | 4 bytes | -2,147,483,647 to 2,147,483,647 |
| float | 4 bytes | +/- 3.4e +/- 38 (7 digits) |
| double | 8 bytes | +/- 1.7e +/- 308 (15 digits) |
| long double | 8 bytes | +/- 1.7e +/- 308 (15 digits) |
| wchar\_t | 2 or 4 bytes | 1 wide character |

Table 2.3: Data type memory and range

Note: We can use a built in function called **sizeof( type )** which takes one of the types listed above and returns the number of bytes in memory used.

**Program 2: Data Type Size**

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter2\_DataTypeSize”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “DataTypeSize.cpp”
5. Replicate program listing 2.

#include <iostream>

int main()

{

std::cout << "Size of char: " << sizeof(char) << " bytes" << std::endl;

std::cout << "Size of int: " << sizeof(int) << " bytes" << std::endl;

std::cout << "Size of short int: " << sizeof(short int) << " bytes" << std::endl;

std::cout << "Size of long int: " << sizeof(long int) << " bytes" << std::endl;

std::cout << "Size of float: " << sizeof(float) << " bytes" << std::endl;

std::cout << "Size of double: " << sizeof(double) << " bytes" << std::endl;

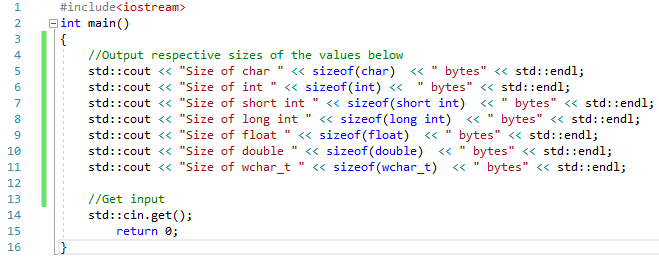
std::cout << "Size of wchar\_t: " << sizeof(wchar\_t) << " bytes" << std::endl;

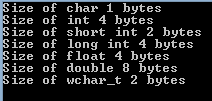
std::cin.get();

return 0;

}

Program Listing 2





**Declaring User Data Types**

Using the keyword **typedef**, you can create data types of your own from the in built types. The reason for doing this is purely for the coders benefit and ease of reading. The compiler reads the new data type as it would the original.

The format for creating your own types is:

typedef type newName;

replace **type** with one of the types detailed in **Table 2.3.**

2.1 Example: Create your own data type to represent the date.

typedef int date;

Now instead of using the in-built type **int** we could use own own user-defined type **date**.

date currentDate;

If however you required a data type to represent several values we can create an **Enumeration**. This allows you to create a new type and assign the constant value it represents. It follows this format:

enum typeName

{

list of names

};

It will become clearer if we use an example. So sticking with the date idea, in this example we would rather use the terms JANUARY, FEBRUARY, MARCH, etc to represent the month. We could create our own **typedef** month as shown above, but then in code we will always be using integer values. This is not very readable, so instead we will use an **enumeration**.

2.2 Example: Create the month enumeration.

enum Month

{

JANUARY = 1,

FEBRUARY,

MARCH,

APRIL,

MAY,

JUNE,

JULY,

AUGUST,

SEPTEMBER,

OCTOBER,

NOVEMBER,

DECEMBER

};

Note: Take note of the syntax used here. After each enumerated type a comma is used except for the last value.

As you can see JANUARY has been given the value of 1. All following names will be incremented by 1. That is to say FEBRUARY will hold the value 2, MARCH the value 3, and so on.

You can set the value of any name, but it is important to remember that the following names will be incremented by 1.

It is also possible to set none of the name to an initial value. In this case the first name will be assigned the value 0, and the following names incremented in the usual way.

To use this enumerated data type we create a variable of this type in the exact same way as we do any other.

Month currentMonth;

Chapter 3: Variable Types

A variable is storage in memory that we can access using a name of our choosing. All variables must be of a data type described in the previous chapter. That is because the data type informs the compiler how much memory is to be reserved and what values can be stored. Refer back to Table 2.1 for the data types that can be used along with the keywords that must entered for C++ to recognise the type.

When defining a variable the user chooses what to name it, but there are restrictions. The name can be composed of letters, digits and underscores, but the first character must always be a letter or underscore. It is good practice to make the first letter of each new word uppercase. For example thisIsMyExampleVariableName.

Also the name chosen should be something informative. Variables of this sort: a, b, or myVariable do not help in any way. Also, there may only ever be one variable of this name. duplicating names will produce syntax errors.

One final thing to note is that C++ is case sensitive, so when using your variables throughout your code ensure that they match. You will receive a syntax error if they don’t.

3.1 Example: Lets create some variables.

The most straightforward way to declare a variable is the following format:

DataType variableName;

This translates in code for a variable called myNum of data type integer to be:

int myNum;

To assign a value to our variable myNum, we use the = operator. This will be covered in more detail in the next chapter.

myNum = 5;

So at this point wherever you use the variable myNum it will produce the value 5. You could however choose to assign the value of 5 at the same time as declaring the variable. This reduces the number of lines of code and make it easier to read. Bare in mind that you will not always know the starting value, so cannot do this.

int myNum = 5;

So far so good. We have one last addition to make to this defining a variable section before we will get on to a mini program, and this is the use of definition lists. We can define multiple variables on the same line as long as they are of the same data type.

double myNum1, myNum2, myNum3;

We can even assign values to these variables as we did with the single variable example above.

double myNum1 = 5.0f, myNum2 = 1.0f, myNum3;

Note: when using **float** it is the practice when assigning values to add an **f** after the value, else the compiler will assume it’s a **double** or **int**. Some compilers will work fine without the **f** but others will give a syntax error.

Note: When assigning a value to a char data type you need to use the single quotes like this ‘a’.

Variables can be changed as often as required. There is no special code required for this, you simply reset the variable to hold a different value.

int myNum = 5;

myNum = 10;

If you require a variable that will never change, then you can use the **const** modifier. This will make the value constant and can never be changed. You may think this unlikely to occur, but it is an feature that you will find useful more often than you may think. It is common to see a lowercase k before the variable name to signify that it is a **const**.

const float kGravity = -9.7f;

const float kPI = 3.14f;

**Program 3: Defining Variables**

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter3\_DefiningVariables”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “DefiningVariables.cpp”
5. Replicate program listing 3.

#include <iostream>

using namespace std;

int main()

{

int myInt1 = 5, myInt2;

myInt2 = 10;

cout << "Value stored in myInt1 is " << myInt1 << endl;

cout << "Value stored in myInt2 is " << myInt2 << endl;

float myFloat1 = 1.2f, myFloat2;

myFloat2= 9.9f;

cout << "Value stored in myFloat1 is " << myFloat1 << endl;

cout << "Value stored in myFloat2 is " << myFloat2 << endl;

char myChar = ‘a’;

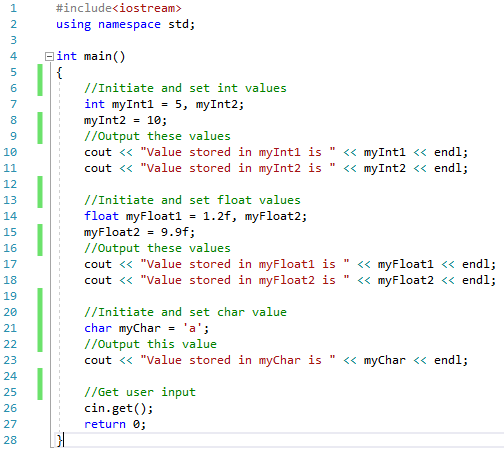
cout << "Value stored in myChar is " << myChar << endl;

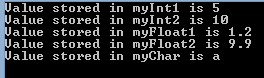
cin.get();

return 0;

}

Program Listing 3



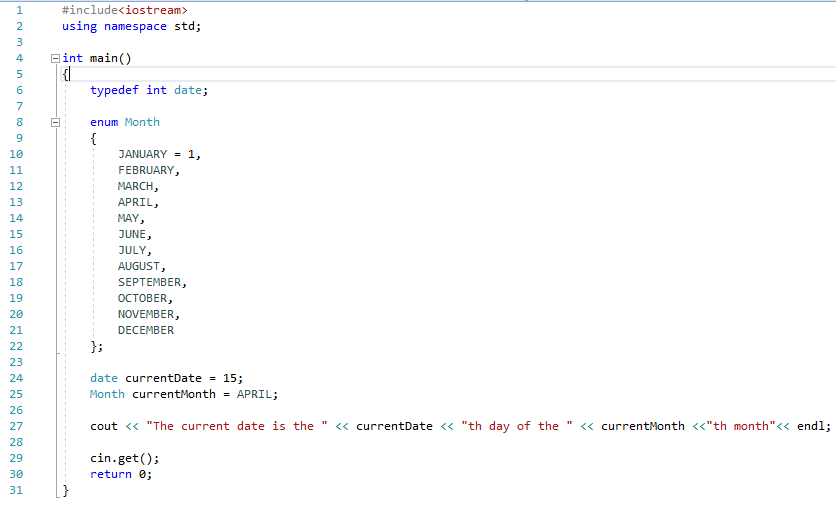


**Program 4: Defining User-Defined Variables**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a program, which creates both a **typedef** data type named date of original data type **int** and an enumerated data type named Month, consisting of the 12 months with JANUARY starting at 1 [See chapter 2].

Next create a variable of data type date, with a starting value of 15, and a variable of data type Month, with its initial value to be APRIL. Output the results to the console screen.





Chapter 4: Operators

Operators are special characters that represent mathematical or logical manipulations. C++ has a range of different operators, and we will be covering the following in this chapter: Mathematical Operators, Assignment Operators, Relational Operators, and Logical Operators.

**Mathematical Operators**

Looking at table 4.1 you will no doubt recognise the first four operators. These represent the usual mathematical operators we use in everyday life. As such we will not be explaining those, instead we will be focusing on the unfamiliar symbols.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Adds two operands |
| - | Subtracts second operand from first |
| \* | Multiplies both operands |
| / | Divides numerator by denominator |
| % | Remainder left after integer division |
| ++ | Increases integer by one |
| -- | Decreases integer by one |

Table 4.1: Mathematical Operators

4.1 Example: Modulus

The following code snippet creates two integer variables which each store a number. The final line of code creates a integer variable named remainder which holds the remaining digits from a b divided by a calculation.

int a = 3;

int b = 10;

int remainder = b % a;

remainder will equal 1 as 3 goes into 10 3 times with a remainder of 1.

4.1 Example: Increment Operator

The increment operator adds 1 to the current value stored in the variable. This is simple enough and will be shown below. Confusion can arise when the increment operator is placed before the variable.

The following code snippet assumes the existence of an integer variable named num, which stores the value 1. **After** the following line of code, num will now be equal to 2.

num++;

The above line of code is equivalent to the following line of code. We assign num to equal its current value plus 1.

num = num + 1;

As stated above, the increment operator can be placed before the variable. This has the affect of changing the value stored in the variable in the same manner as before. But the effect takes place immediately.

++num;

Assuming num started with a value of 1, in the above line of code, num will equal 2 as soon as it is reached, not on the following line.

Note: The placement of the Increment Operator may seem trivial or confusing at this point in time, but once we get to using loops in chapter 6 the placement will be crucial.

4.2 Example: Decrement Operator

The decrement operator subtracts 1 from the current value stored in the variable. It works very much like the increment operator.

Again, the following code snippet assumes the existence of an integer variable named num, which stores the value 1. **After** the following line of code, num will now be equal to 0.

num--;

The above line of code is equivalent to the following line of code. We assign num to equal its current value minus 1.

num = num - 1;

As stated above, the increment operator can be placed before the variable. This has the affect of changing the value stored in the variable in the same manner as before. But the effect takes place immediately.

--num;

Assuming num started with a value of 1, in the above line of code, num will equal 0 as soon as it is reached, not on the following line.

**Program 5: Increment / Decrement Operators**

1. To begin, start Visual Studio.
2. Create a new project via File -> New -> Project or Ctrl+Shift+N Name it “Chapter4\_IncrementDecrementOperators”
3. Click **Next** and you should be greeted with the following screen. Make sure to have **Empty Project** ticked and click **Finish**.
4. Add a new source file and name it “IncrementDecrementOperators.cpp”
5. Replicate program listing 5.

Note: It is a good idea to comment your code. This means to add comments, which are only there for the programmers benefit. The compiler ignores them. This may seem trivial, but it really helps other programmers who may use your code, or even yourself if you haven’t looked at a particular program in a while. To comment you can either use a double slash // or surround the text with a /\* \*/ block. Both are demonstrated in the following code.

#include <iostream>

using namespace std;

int main()

{

// Initialise num to 1

int num = 1;

cout << “num = “ << num << endl;

cout << “Increment Operator before num = “ << ++num << endl;

cout << “num after = “ << num << endl;

/\* Reset num to 1

This style of comment can span multiple lines \*/

num = 1;

cout << “num = “ << num << endl;

cout << “Increment Operator after num = “ << num++ << endl;

cout << “num after = “ << num << endl;

// Reset num to 1

num = 1;

cout << “num = “ << num << endl;

cout << “Decrement Operator after num = “ << --num << endl;

cout << “num after = “ << num << endl;

/\* Reset num to 1 \*/

num = 1;

cout << “num = “ << num << endl;

cout << “Decrement Operator after num = “ << num-- << endl;

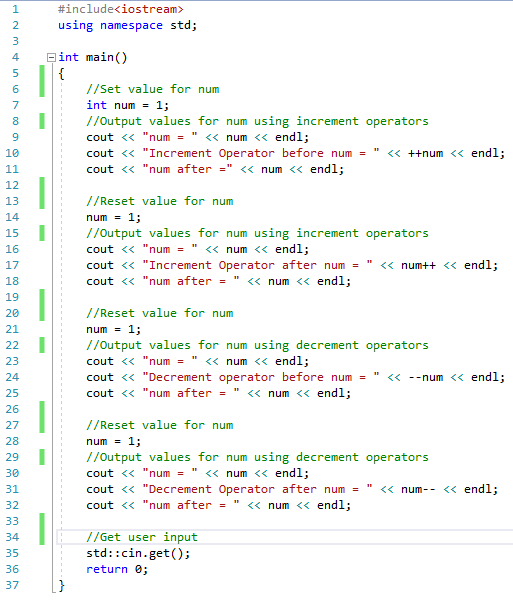
cout << “num after = “ << num << endl;

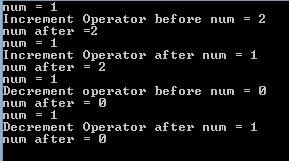
std::cin.get();

return 0;

}

Program Listing 5





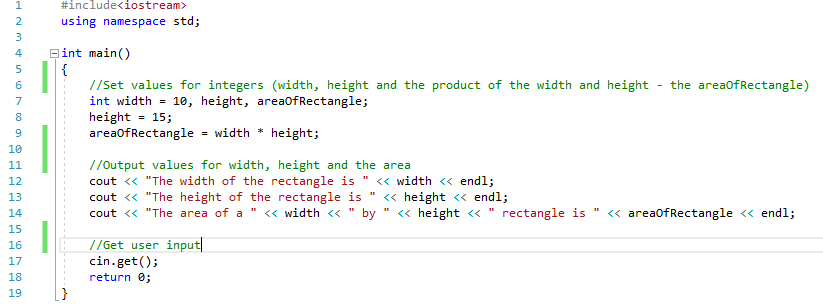
**Program 6: Area of a Rectangle**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a program, which creates variables of integer data type for **width** & **height**. Initialise width with a starting value of 10 and height to 15. Next create an integer variable named **areaOfRectangle** and set this to equal the width times by the height.

Output the result in the following format:

cout << “The area of a “ << width << “ by “ << height << “ rectangle is “ << areaOfRectangle << endl;





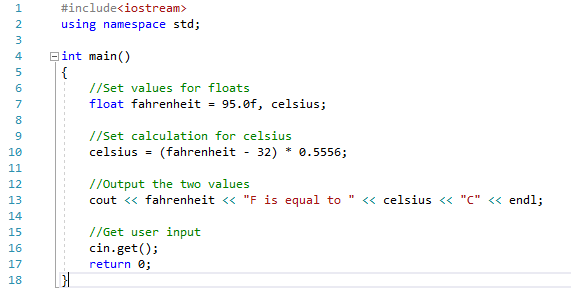
**Program 7: Converting from Fahrenheit to Celsius**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a program, which creates two variables of floating point data type named **fahrenheit** and **celsius**. Set the starting value of **fahrenheit** to equal 95.0f. Calculate **celsius** using the following formula: Celsius = (Fahrenheit – 32) \* 0.5556.

Output the result in the following format:

cout << fahrenheit << “F is equal to “ << celsius << “C” << endl;





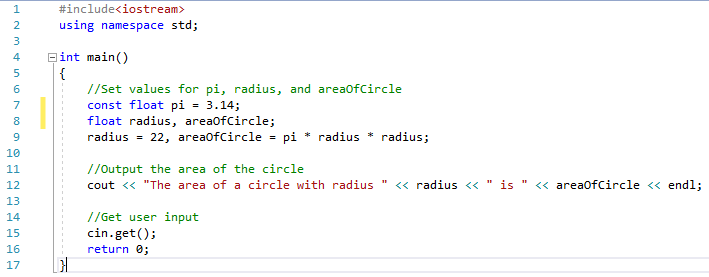
**Program 8: Area of a Circle**

This is for you to complete. You need to add the code listing and screen shot for this program here. It is crucial that you do this to keep your portfolio as a useful resource as the end of year test will require a selection of these programs.

Write a program, which creates a constant variable of float type named pi. This will hold the value 3.14. Next create two variables of floating point data type named **radius** and **areaOfCircle**. Set the starting value of r**adius** to equal 22.0f. Calculate **areaOfCircle** using the following formula: Area of a circle = pi \* radius \* radius.

Output the result in the following format:

cout << “The area of a circle with radius “ << radius << “ is ” << areaOfCircle << endl;





**Assignment Operators**

Now that we have an understanding of the mathematical operators, we can move on to assignment operators. These operators simply reduce the amount of code required to do a simple operation. Take a look at Table 4.2. Examples of each operator will follow below.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Assigns value from the right side operand to the left side operand. |
| += | Adds right operand to the left operand and assigns the result to the left operand. |
| -= | Subtracts right operand to the left operand and assigns the result to the left operand. |
| \*= | Multiplies the right and left operands together and assigns the result to the left operand. |
| /= | Divides the left operand by the right operand and assigns the result to the left operand. |
| %= | Takes the modulus of two operands and assigns the result to the left operand. |

Table 4.2: Assignment Operators

4.3 Example: Equals Operator

As has been shown previously the equals operator is used to assign a value to a variable. A variable must always be on the left hand side of these operations.

The following snippet assigns the value of 5 to the variable num.

int num = 5;

You can also assign a variable to equal the value stored in a completely different variable as long as they are of the same data type.

int num1 = 5;

int num2 = num1;

The following code snippets are both invalid and will result in syntax errors.

5 = num;

5 = 10;

4.4 Example: Add AND Operator

num1 += num2;

is equivalent to:

num1 = num1 + num2;

4.5 Example: Subtract AND Operator

num1 -= num2;

is equivalent to:

num1 = num1 - num2;

4.6 Example: Multiply AND Operator

num1 \*= num2;

is equivalent to:

num1 = num1 \* num2;

4.7 Example: Divide AND Operator

num1 /= num2;

is equivalent to:

num1 = num1 / num2;

4.8 Example: Modulus AND Operator

num1 %= num2;

is equivalent to:

num1 = num1 % num2;

**Relational Operators**

Relational Operators are used to return a result of true or false. These will be used all the time in conjunction with conditionals [see chapter 5]. Take a look at Table 4.3 and the corresponding examples below for an explanation of each operator.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Checks if the values of the two operands are equal, if so the condition becomes true. |
| != | Checks if the values of the two operands are not equal, if hey are not equal then the condition becomes true. |
| > | Checks if the left operand is greater than the right operand, if so the condition becomes true. |
| < | Checks if the left operand is less than the right operand, if so the condition becomes true. |
| >= | Checks if the left operand is greater than or equal to the right operand, if so the condition becomes true. |
| <= | Checks if the left operand is less than or equal to the right operand, if so the condition becomes true. |

Table 4.3: Relational Operators

4.9 Example: Is Equal To Operator

If the variables being compared store the same value the result returned will be true, otherwise it will return false. Take notice of how a single = sign is for assignment and the double == is for comparison.

bool isEqual = ( num1 == num2 );

4.10 Example: Does Not Equal Operator

If the variables being compared store different values the result returned will be true, otherwise it will return false.

bool doesNotEqual = ( num1 != num2 );

4.11 Example: Greater Than Operator

If the left hand variable is greater than the right hand variable the result returned will be true, otherwise false is returned.

bool greaterThan = ( num1 > num2 );

4.12 Example: Less Than Operator

If the left hand variable is less than the right hand variable the result returned will be true, otherwise false is returned.

bool lessThan = ( num1 < num2 );

4.13 Example: Greater Than or Equal To Operator

If the left hand variable is greater than or the same as the right hand variable the result returned will be true, otherwise false is returned.

bool greaterThanOrEqualTo = ( num1 >= num2 );

4.14 Example: Less Than or Equal To Operator

If the left hand variable is less than or the same as the right hand variable the result returned will be true, otherwise false is returned.

bool lessThanOrEqualTo = ( num1 <= num2 );

**Logical Operators**

Logical Operators are used to return a result of true or false. Just like Relational Operators, these will be used all the time in conjunction with conditionals [see chapter 5]. Take a look at Table 4.4 and the corresponding examples below for an explanation of each operator.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | AND Operator  If both operands are non-zero [true], then the condition becomes true. |
| || | OR Operator  If any of the two operands is non-zero [true], then condition becomes true. |
| ! | NOT Operator  Used to reverse the logical state of an operand. If a condition is true, the NOT Operator will make it false. |

Table 4.4: Logical Operators

4.15 Example: Logical AND Operator

Logical AND checks if both variables being checked are true the result will be true, otherwise false will be returned.

This example will set bothTrue to true.

bool a = true;

bool b = true;

bool bothTrue = ( a && b );

This example will set bothTrue to false as b now equals false.

bool a = true;

bool b = false;

bool bothTrue = ( a && b );

4.16 Example: Logical OR Operator

Logical OR checks if either of the variables being checked are true. If one of them is true then the result returned will be true, otherwise false will be returned.

This example will set eitherTrue to true as a is set to true.

bool a = true;

bool b = false;

bool eitherTrue = ( a || b );

This example will set eitherTrue to false as both a and b now equal false.

bool a = false;

bool b = false;

bool eitherTrue = ( a || b );

4.17 Example: Logical NOT Operator

The Logical NOT operator will reverse the result from the other two logical operators. This may seem a redundant operation, but using the ! (NOT) operator will come in useful. The following two examples are the exact same as the examples used for Logical AND (Example 4.15) except that we have included the ! (NOT) operator. Notice how the results have been reversed.

This example sets notBothTrue to false as both a and b are set to true.

bool a = true;

bool b = true;

bool notBothTrue = !( a && b );

This example sets notBothTrue to true as both b is now equalt to false meaning that both are no longer set to true.

bool a = true;

bool b = false;

bool notBothTrue = !( a && b );