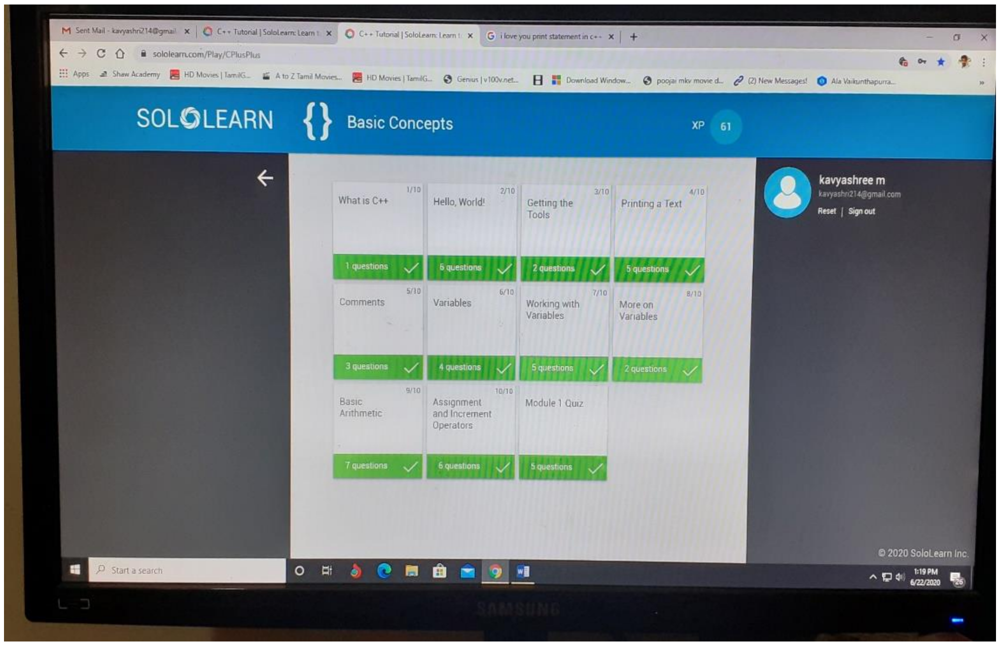
DAILY ASSESSMENT

Date:	**22-6-2020**	**Name:**	**Kavyashree m**
Course:	**C++**	**USN:**	**4al15ec036**
Topic:	**What is c++ ?, hello world, getting the tools, printing a text, comments, variables, working with variables, more on variables, basic arithmetic, assignment and increment operator.**	**Semester & Section:**	**8th A**
GitHub Repository:	**kavya**		

FORENOON SESSION DETAILS


SOLOLEARN
Basic Concepts
XP 61
kavyashree m
kavyashri214@gmail.com
Reset | Sign out
What is C++ 1/10 1 questions
Hello, World! 2/10 6 questions
Getting the Tools 3/10 2 questions
Printing a Text 4/10 5 questions
Comments 5/10 3 questions
Variables 6/10 4 questions
Working with Variables 7/10 5 questions
More on Variables 8/10 2 questions
Basic Arithmetic 9/10 7 questions
Assignment and Increment Operators 10/10 6 questions
Module 1 Quiz 5 questions
© 2020 SoloLearn Inc.

What is c++?

C++ is a general-purpose programming language.

C++ is used to create computer programs.

Anything from art applications, music players and even video games!

C++ was derived from C, and is largely based on it.

Hello world

Below is a simple code that has "Hello world!" as its output.

```
#include<iostream>
using namespace std;
int main ()
{
cout << "Hello world!";
return 0;
}
```

C++ offers various headers, each of which contains information needed for programs to work properly. This particular program calls for the header <iostream>. The number sign (#) at the beginning of a line targets the compiler's pre-processor. In this case, #include tells the pre-processor to include the <iostream> header. The <iostream> header defines the standard stream objects that input and output data.

In general, blank lines serve to improve the code's readability and structure. Whitespace, such as spaces, tabs, and newlines, is also ignored, although it is used to enhance the program's visual attractiveness.

```
#include<iostream>
using namespace std;
int main()
{
cout << "Hello world!";
return 0;
}
```

In our code, the line using namespace std; tells the compiler to use the std (standard) namespace.
The std namespace includes features of the C++ Standard Library.

Main

Program execution begins with the main function, int main().

```
#include<iostream>
using namespace std;

int main()
{
cout << "Hello world!";
return0;
}
```

Curly brackets { } indicate the beginning and end of a function, which can also be called the function's body. The information inside the brackets indicates what the function does when executed.
The entry point of every C++ program is main(), irrespective of what the program. The next line, cout << "Hello world!"; results in the display of "Hello world!" to the screen.

```
#include<iostream>
using namespace std;
int main()
```

```
{
cout << "Hello world!";
return 0;
}
```

In C++, streams are used to perform input and output operations. In most program environments, the standard default output destination is the screen. In C++, cout is the stream object used to access it. cout is used in combination with the insertion operator. Write the insertion operator as << to insert the data that comes after it into the stream that comes before. In C++, the semicolon is used to terminate a statement. Each statement must end with a semicolon. It indicates the end of one logical expression.

Statements

A block is a set of logically connected statements, surrounded by opening and closing curly braces.

For example:

```
{
cout << "Hello world!";
return 0;
}
```

You can have multiple statements on a single line, as long as you remember to end each statement with a semicolon. Failing to do so will result in an error.

Return

The last instruction in the program is the return statement. The line return 0; terminates the main () function and causes it to return the value 0 to the calling process. A non-zero value (usually of 1) signals abnormal termination.

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello world!";
return 0;
}
```

If the return statement is left off, the C++ compiler implicitly inserts "return 0;" to the end of the main () function.

Getting the Tools

You can run, save, and share your C++ codes on our Code Playground, without installing any additional software.

You need both of the following components to build C++ programs.

1. Integrated Development Environment (IDE): Provides tools for writing source code. Any text editor can be used as an IDE.
2. Compiler: Compiles source code into the final executable program. There are a number of C++ compilers available. The most frequently used and free available compiler is the GNU C/C++ compiler.

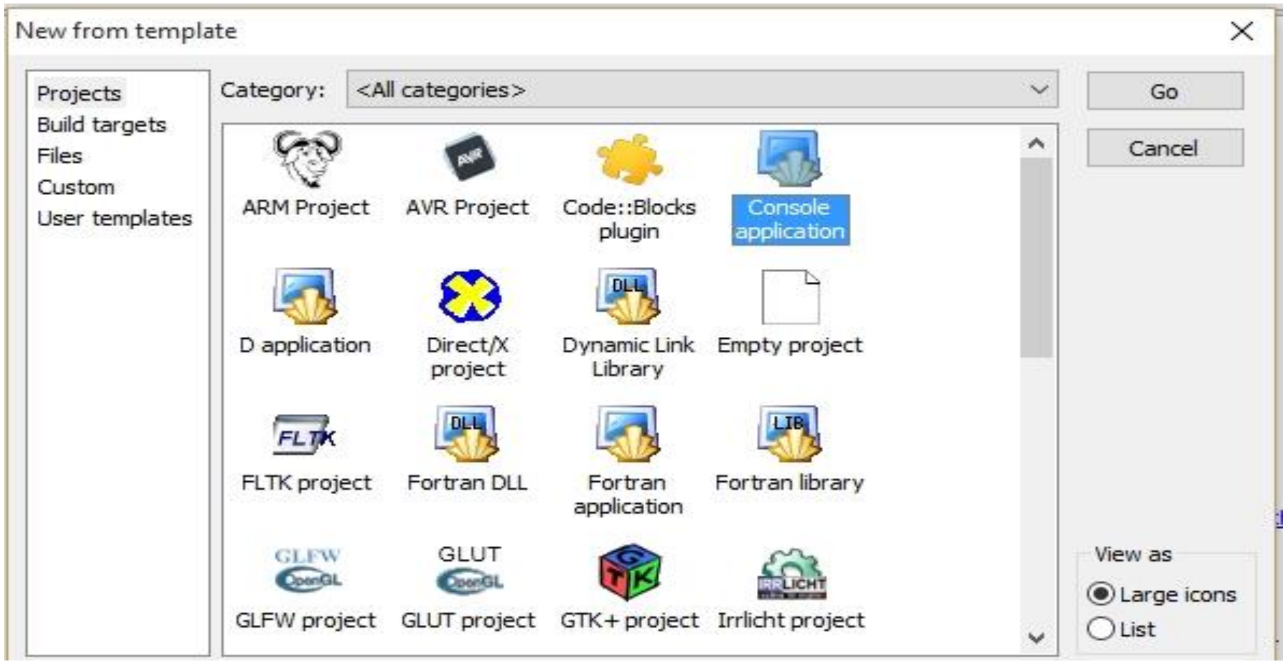
Various C++ IDEs and compilers are available. We'll use a free tool called Code::Blocks, which includes both an IDE and a compiler, and is available for Windows, Linux and MacOS.

To download Code::Blocks, go to http://www.codeblocks.org/, Click the Downloads link, and choose "Download the binary release". Choose your OS and download the setup file, which includes the C++ compiler (For Windows, it's the one with mingw in the name). Make sure to install the version that includes the compiler.

Getting the Tools

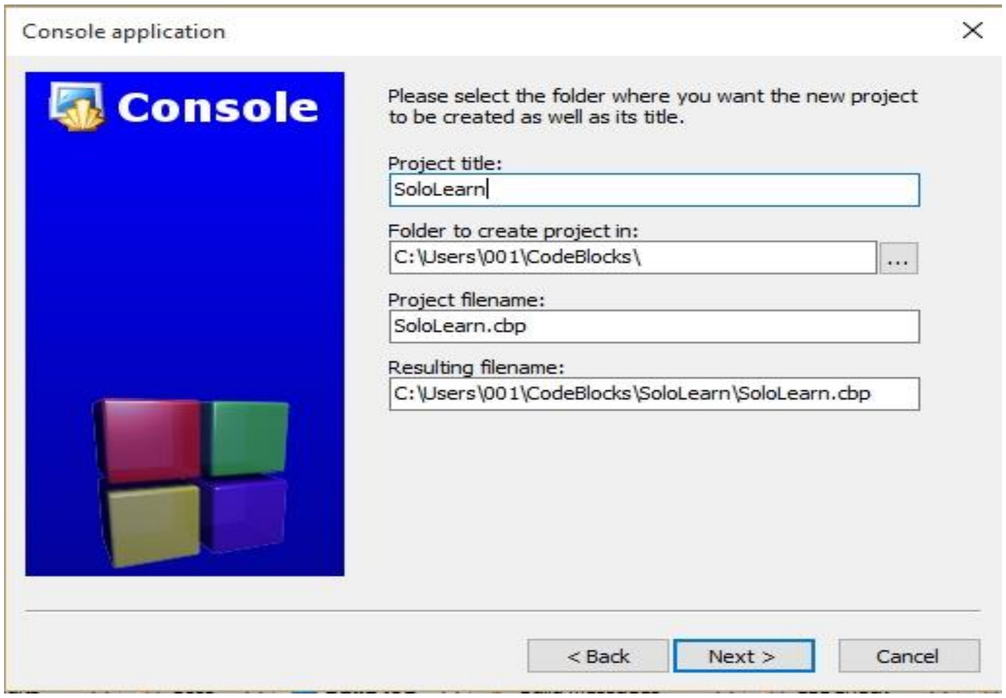
To create a project, open Code::Blocks and click "Create a new project" (or File->New->Project).

This will open a dialog of project templates. Choose Console application and click Go.



Go through the wizard, making sure that C++ is selected as the language.

Give your project a name and specify a folder to save it to.



Please select the folder where you want the new project to be created as well as its title.

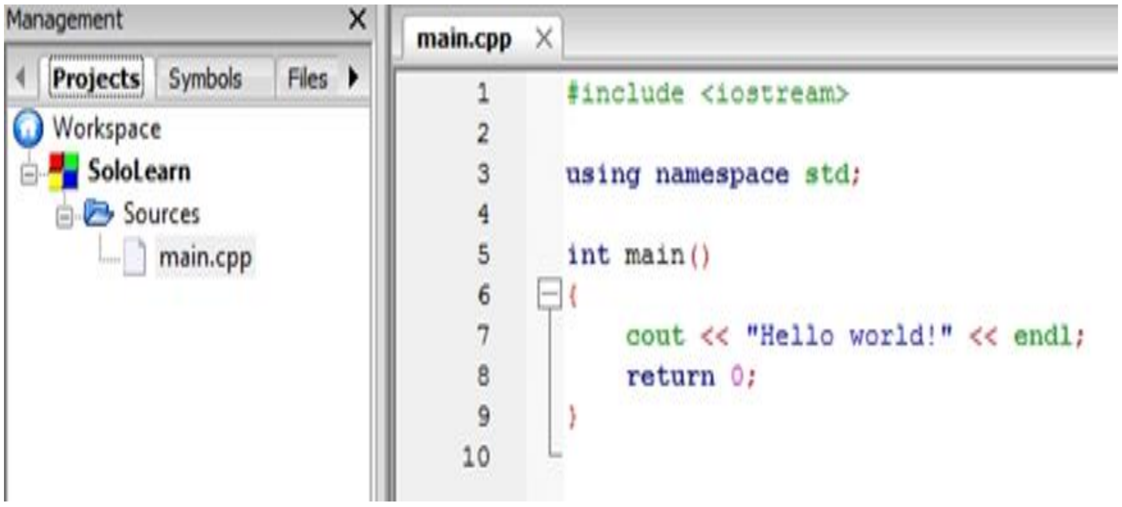
Project title: SoloLearn

Folder to create project in: C:\Users\001\CodeBlocks\

Project filename: SoloLearn.cbp

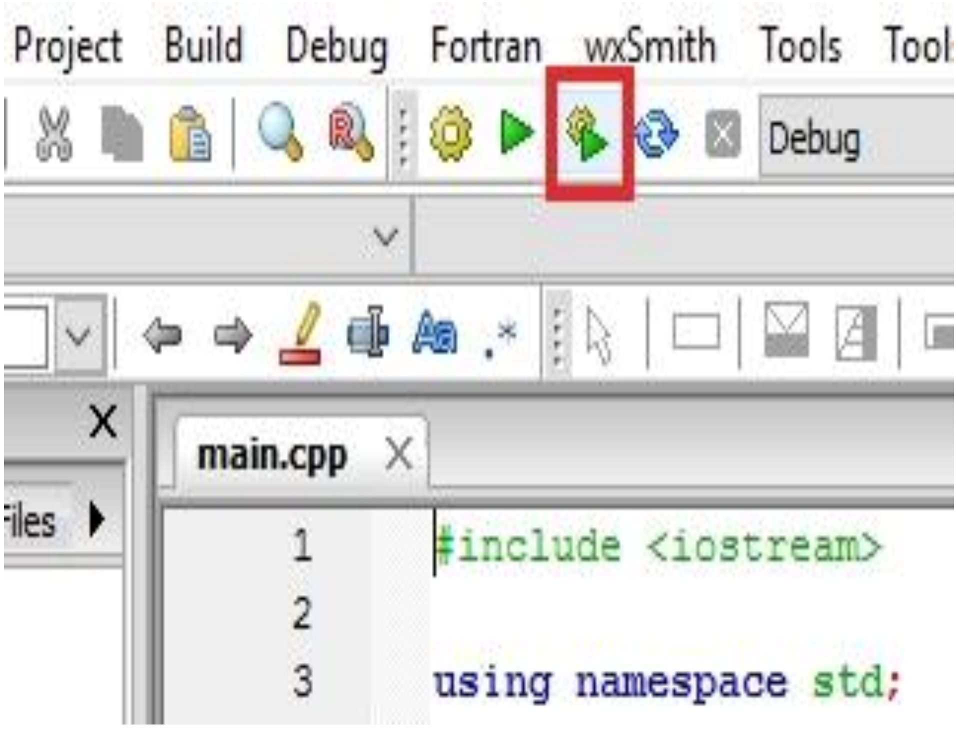
Resulting filename: C:\Users\001\CodeBlocks\SoloLearn\SoloLearn.cbp

Make sure the Compiler is selected, and click Finish. GNU GCC is one of the popular compilers available for Code::Blocks. On the left sidebar, expand Sources. You'll see your project, along with its source files. Code::Blocks automatically created a main.cpp file that includes a basic Hello World program (C++ source files have .cpp, .cp or .c extensions).



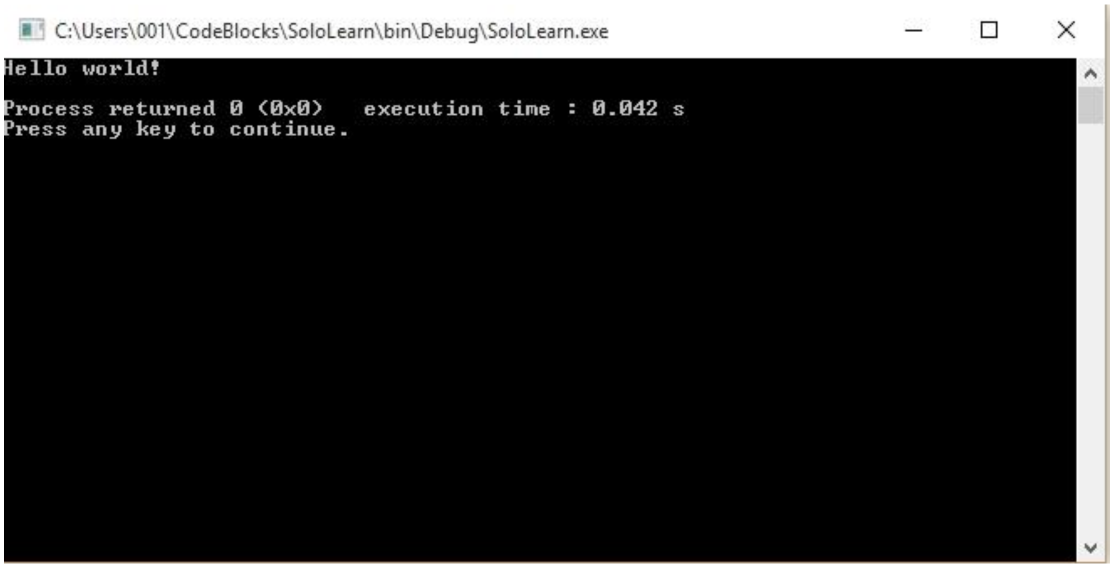
```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Click on the "Build and Run" icon in the toolbar to compile and run the program.



```
#include <iostream>

using namespace std;
```

A console window will open and display program output.



```
Hello world!

Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

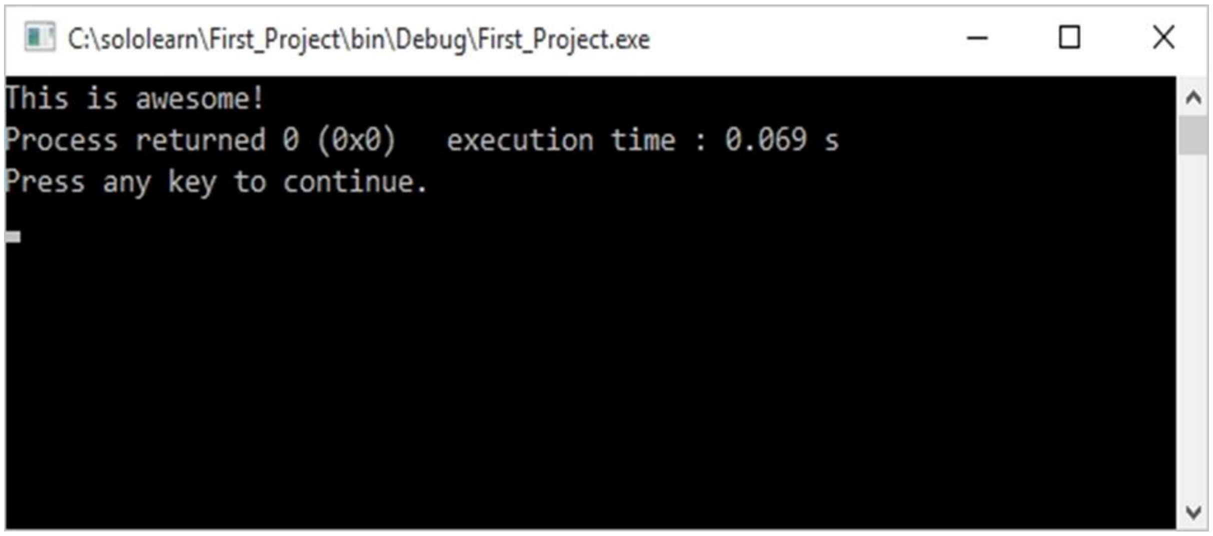
You can run, save, and share your C++ codes on our Code Playground, without installing any additional software.

Your First C++ Program

You can add multiple insertion operators after cout.

cout << "This " << "is " << "awesome!";

Result:



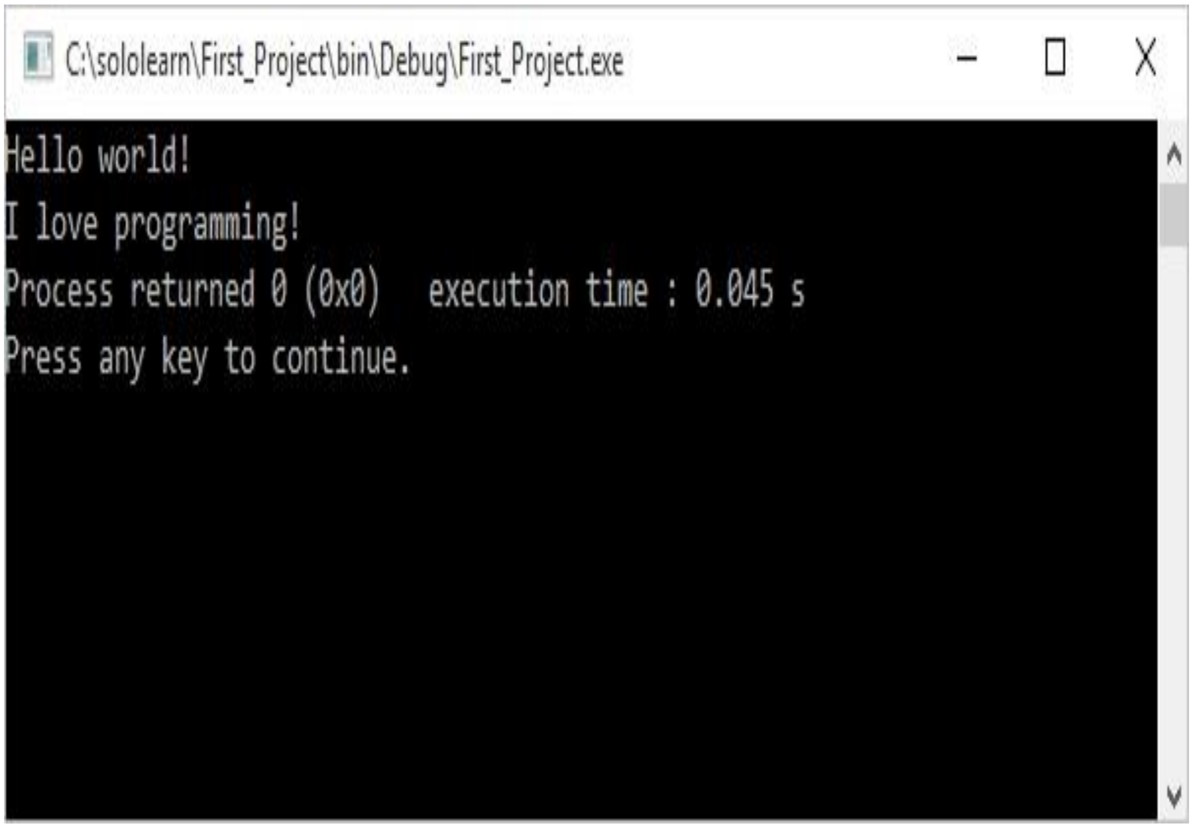
```
This is awesome!
Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

New Line

The cout operator does not insert a line break at the end of the output. One way to print two lines is to use the endl manipulator, which will put in a line break.

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello world!" << endl;
cout << "I love programming!";
return 0;
}
```

The endl manipulator moves down to a new line to print the second text.


C:\sololearn\First_Project\bin\Debug\First_Project.exe

```
Hello world!
I love programming!
Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

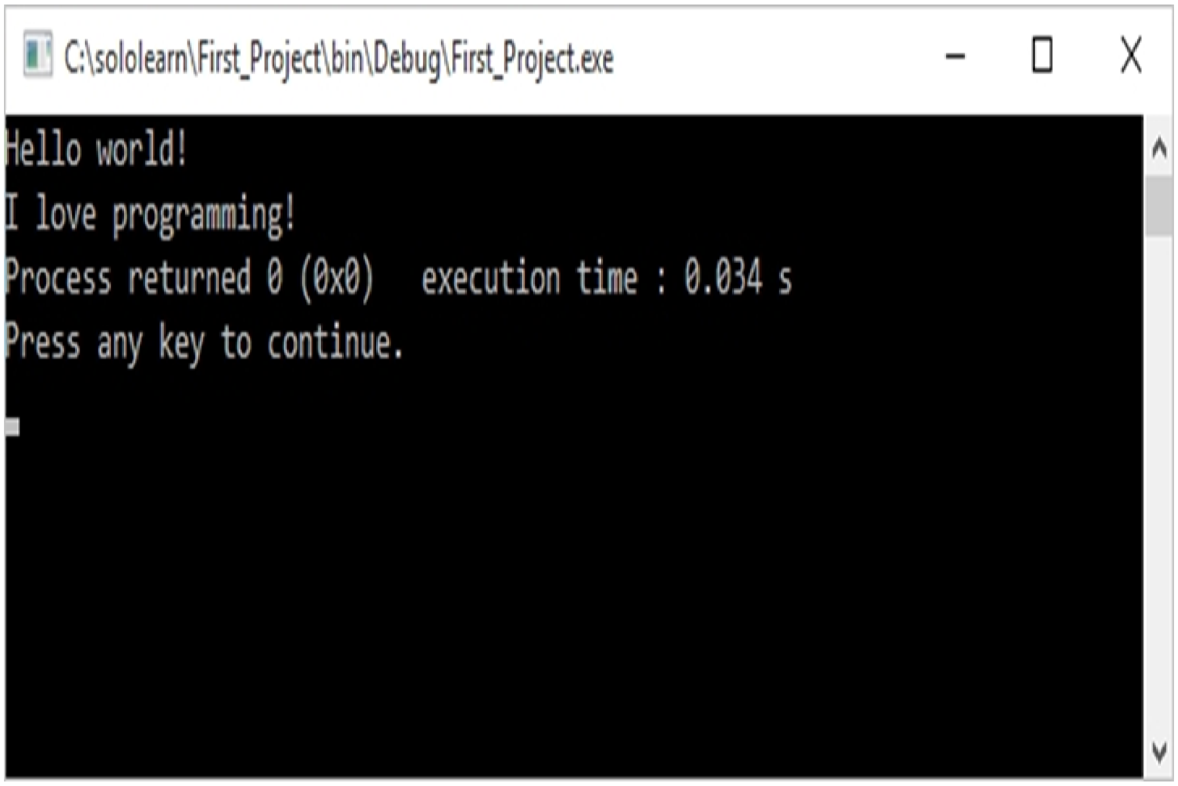
The new line character \n can be used as an alternative to endl. The backslash (\) is called an escape character, and indicates a "special" character.

Example:

```
#include<iostream>
using namespace std;
int main()
{
cout << "Hello world! \n";
cout << "I love programming!";
return 0;
```

}

Result:



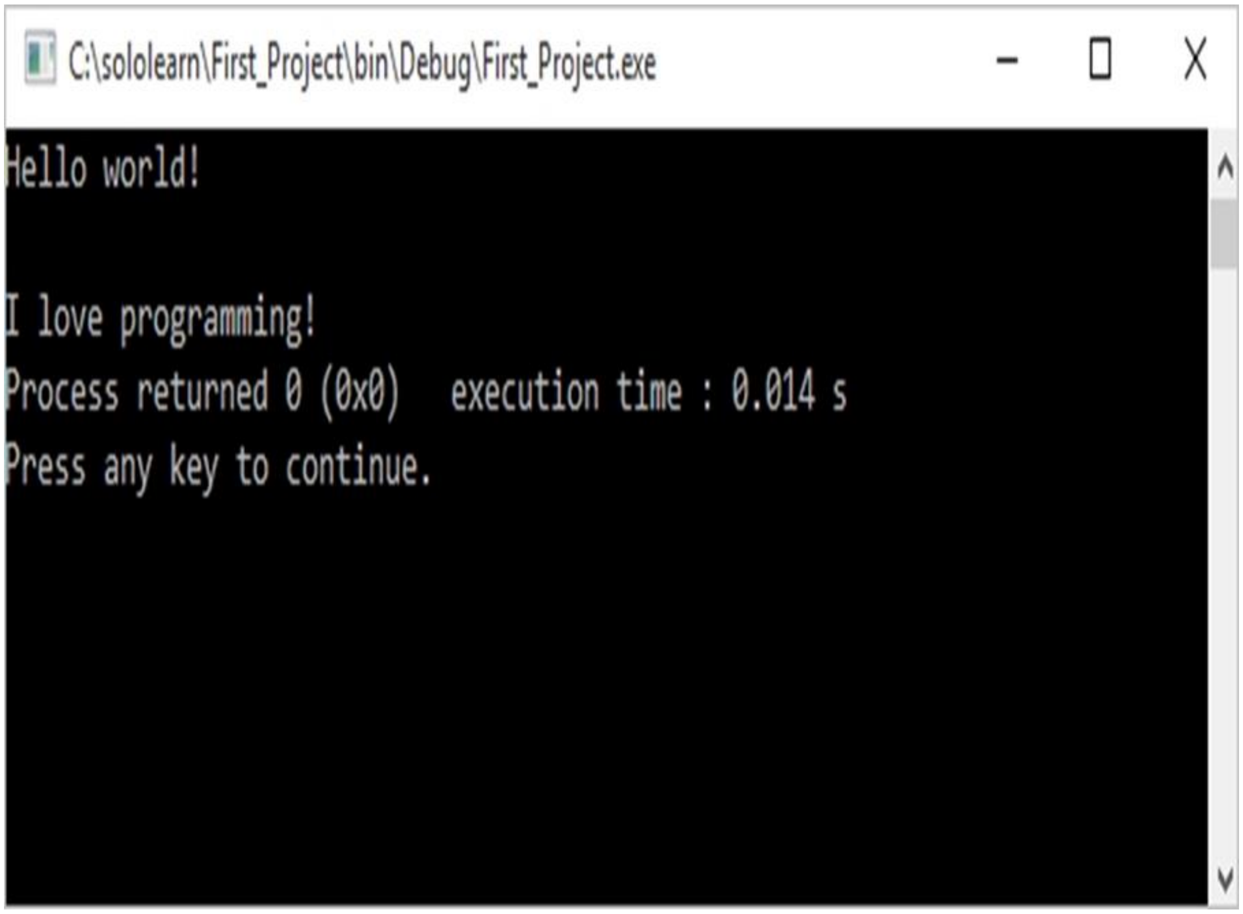
C:\sololearn\First_Project\bin\Debug\First_Project.exe

```
Hello world!
I love programming!
Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```

Two newline characters placed together result in a blank line.

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello world! \n\n";
cout << "I love programming!";
return 0;
}
```

Result:



```
C:\sololearn\First_Project\bin\Debug\First_Project.exe
Hello world!

I love programming!
Process returned 0 (0x0)   execution time : 0.014 s
Press any key to continue.
```

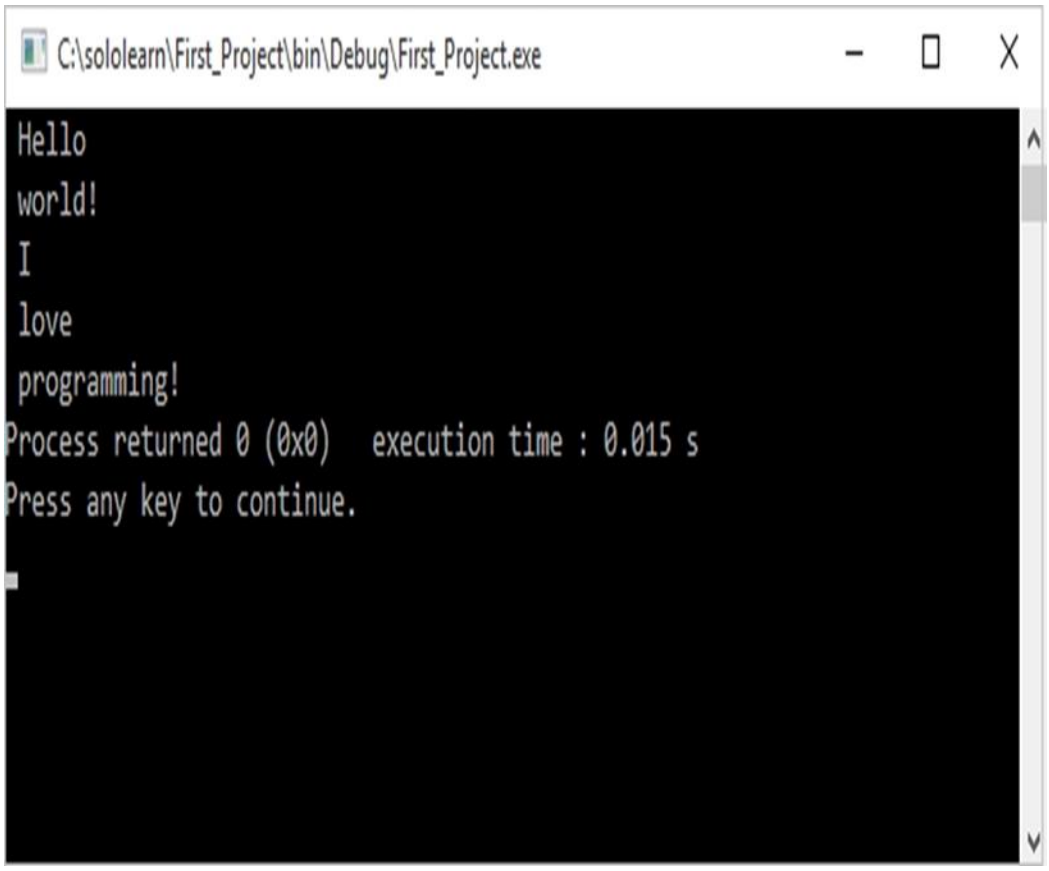
Multiple New Lines

Using a single cout statement with as many instances of \n as your program requires will print out multiple lines of text.

```
#include <iostream>
using namespace std;
int main()
{
cout << " Hello \n world! \n I \n love \n programming!";
return 0;
```

}

Result:


C:\sololearn\First_Project\bin\Debug\First_Project.exe

```
Hello
world!
I
love
programming!
Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

Comments

Comments are explanatory statements that you can include in the C++ code to explain what the code is doing. The compiler ignores everything that appears in the comment, so none of that information shows in the result.A comment beginning with two slashes (//) is called a single-line comment.

For example:

```
#include <iostream>
using namespace std;
int main()
```

{

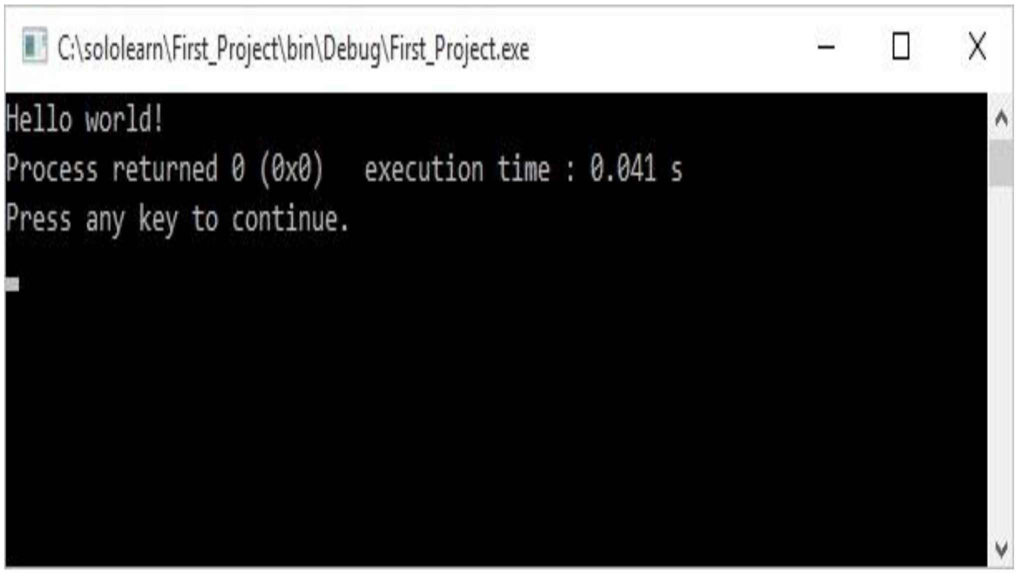
// prints "Hello world"

cout << "Hello world!";

return 0;

}

When the above code is compiled, it will ignore the // prints "Hello world" statement and will produce the following result:



```
Hello world!
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

Multi-Line Comments

Comments that require multiple lines begin with /* and end with */

You can place them on the same line or insert one or more lines between them.

/* This is a comment */

/* C++ comments can span multiple lines*/

Using Comments

Comments can be written anywhere, and can be repeated any number of times throughout the code. Within a comment marked with /* and */, // characters have no special meaning, and vice versa. This allows you to "nest" one comment type within the other.

```
/* Comment out printing of Hello world!
cout << "Hello world!"; // prints Hello world! */
```

Variables

Creating a variable reserves a memory location, or a space in memory for storing values. The compiler requires that you provide a data type for each variable you declare. C++ offer a rich assortment of built-in as well as user defined data types.

Integer, a built-in type, represents a whole number value. Define integer using the keyword int. C++ requires that you specify the type and the identifier for each variable defined. An identifier is a name for a variable, function, class, module, or any other user-defined item. An identifier starts with a letter (A-Z or a-z) or an underscore (_), followed by additional letters, underscores, and digits (0 to 9).

For example, define a variable called myVariable that can hold integer values as follows:

```
int myVariable = 10;
```

Now, let's assign a value to the variable and print it.

```
#include <iostream>
using namespace std;
int main()
{
int myVariable = 10;
cout << myVariable;
return 0;
}
// Outputs 10
```

Variables

Define all variables with a name and a data type before using them in a program. In cases in which you have multiple variables of the same type, it's possible to define them in one declaration, separating them with commas.int a, b;

```
// defines two variables of type int
```

A variable can be assigned a value, and can be used to perform operations. For example, we can create an additional variable called sum, and add two variables together.

```
int a = 30;
int b = 15;
int sum = a + b;
// Now sum equals 45
```

Let's create a program to calculate and print the sum of two integers.

```
#include <iostream>
using namespace std;
int main()
{
int a = 30;
int b = 12;
int sum = a + b;
cout << sum;
return 0;
}
//Outputs 42
```

Declaring Variables

You have the option to assign a value to the variable at the time you declare the variable or to declare it and assign a value later. You can also change the value of a variable. Some examples:

```
int a;
int b = 42;
a = 10;
b = 3;
```

User Input

To enable the user to input a value, use cin in combination with the extraction operator (>>). The variable containing the extracted data follows the operator.

The following example shows how to accept user input and store it in the num variable:

```
int num;
cin >> num;
```

Accepting User Input

The following program prompts the user to input a number and stores it in the variable a:

```
#include <iostream>
using namespace std;
int main()
{
int a;
cout << "Please enter a number \n";
cin >> a;
return 0;
}
```

When the program runs, it displays the message "Please enter a number", and then waits for the user to enter a number and press Enter, or Return. The entered number is stored in the variable a.

Let's create a program that accepts the input of two numbers and prints their sum.

```
#include<iostream>
using namespace std;
int main()
{
int a, b;
int sum;
cout << "Enter a number \n";
cin >> a;
cout << "Enter another number \n";
cin >> b;
sum = a + b;
cout << "Sum is: " << sum << endl;
return 0;
}
```

Specifying the data type is required just once, at the time when the variable is declared. After that, the variable may be used without referring to the data type.

```
int a;
a = 10;
```

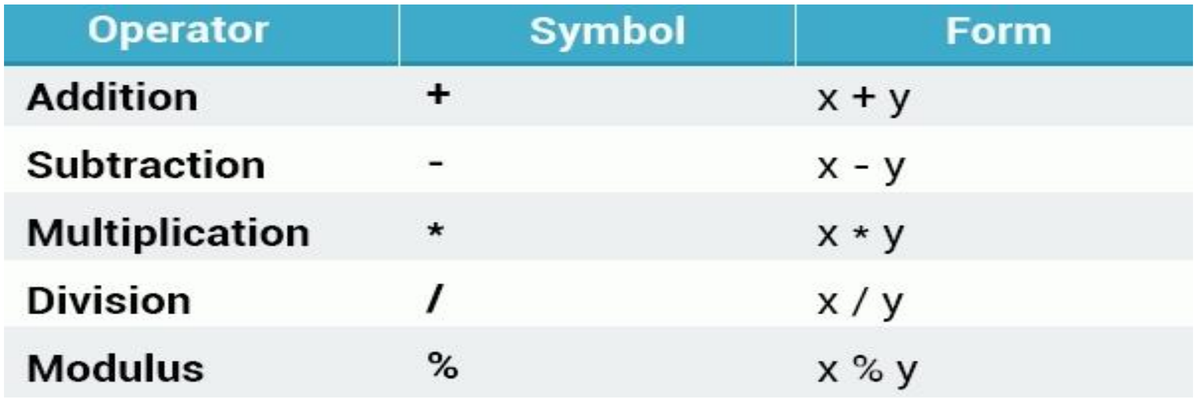
A variable's value may be changed as many times as necessary throughout the program.

For example:

```
int a = 100;

a = 50;

cout << a;

// Outputs 50
```

Arithmetic Operators

C++ supports these arithmetic operators.



Operator	Symbol	Form
Addition	+	x + y
Subtraction	-	x - y
Multiplication	*	x * y
Division	/	x / y
Modulus	%	x % y

The addition operator adds its operands together.

```
int x = 40 + 60;

cout << x;

// Outputs 100
```

Subtraction

The subtraction operator subtracts one operand from the other.

```
int x = 100 - 60;

cout << x;
```

```
//Outputs 40
```

Multiplication

The multiplication operator multiplies its operands.

```
int x = 5 * 6;
cout << x;
//Outputs 30
```

Division

The division operator divides the first operand by the second. Any remainder is dropped in order to return an integer value.

Example:

```
int x = 10 / 3;
cout << x;
// Outputs 3
```

If one or both of the operands are floating point values, the division operator performs floating point division.

Dividing by 0 will crash your program.

Modulus

The modulus operator (%) is informally known as the remainder operator because it returns the remainder after an integer division.

For example:

```
int x = 25 % 7;
cout << x;
```

// Outputs 4

Operator Precedence

Operator precedence determines the grouping of terms in an expression, which affects how an expression is evaluated. Certain operators take higher precedence over others; for example, the multiplication operator has higher precedence over the addition operator.

For example:

```
int x = 5+2*2;
cout << x;
// Outputs 9
```

The program above evaluates 2*2 first, and then adds the result to 5. As in mathematics, using parentheses alters operator precedence.

```
int x = (5 + 2) *2;cout << x;
// Outputs 14
```

Parentheses force the operations to have higher precedence. If there are parenthetical expressions nested within one another, the expression within the innermost parentheses is evaluated first.

If none of the expressions are in parentheses, multiplicative (multiplication, division, modulus) operators will be evaluated before additive (addition, subtraction) operators

Assignment Operators

The simple assignment operator (=) assigns the right side to the left side. C++ provides shorthand operators that have the capability of performing an operation and an assignment at the same time.

For example:

int x = 10;

x += 4; // equivalent to x = x + 4

x -= 5; // equivalent to x = x – 5

The same shorthand syntax applies to the multiplication, division, and modulus operators’

*= 3; // equivalent to x = x * 3

x /= 2; // equivalent to x = x / 2

x %= 4; // equivalent to x = x % 4

x /= 2; // equivalent to x = x / 2

x %= 4; // equivalent to x = x % 4

The increment operator is used to increase an integer's value by one, and is a commonly used C++ operator.

x++; //equivalent to x = x + 1

Increment Operator

The increment operator has two forms, prefix and postfix.

++x; // prefix

x++; // postfix

Prefix increments the value, and then proceeds with the expression. Postfix evaluates the expression and then performs the incrementing.

Prefix example:

x = 5;

y = ++x;

// x is 6, y is 6

Postfix example:

```
x = 5;
y = x++;
// x is 6, y is 5
```

Decrement Operator

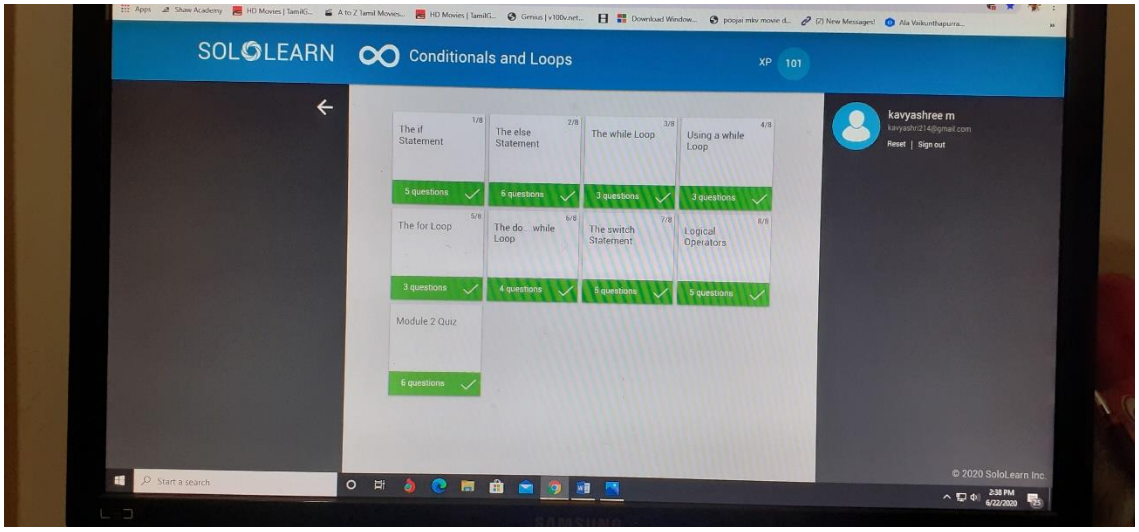
The decrement operator (--) works in much the same way as the increment operator, but instead of increasing the value, it decreases it by one.

```
--x; // prefix
x--; // postfix
```

AFTERNOON SESSION DETAILS

Date:	**22-6-2020**	**Name:**	**Kavyashree m**
Course:	**C++**	**USN:**	**4al15ec036**
Topic:	**The if statement, the else statement , the while loop, using a while loop , the for loop , the do….while loop ,the switch statement , the logical operators**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

Image of session



The if statement

The if statement is used to execute some code if a condition is true. Syntax:

if (condition) {

statements

}

The condition specifies which expression is to be evaluated. If the condition is true, the statements in the curly brackets are executed.

Then if Statement

Use relational operators to evaluate conditions.

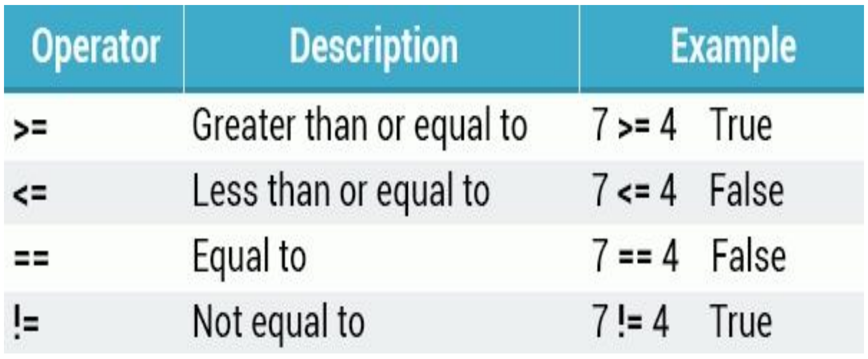
For example:

```
if (7 > 4) {
cout << "Yes";
}
// Outputs "Yes"
```

The if statement evaluates the condition (7>4), finds it to be true, and then executes the cout statement. If we change the greater operator to a less than operator (7<4), the statement will not be executed and nothing will be printed out. A condition specified in an if statement does not require a semicolon.

Relational Operators

Additional relational operators:



Operator	Description	Example
>=	Greater than or equal to	7 >= 4 True
<=	Less than or equal to	7 <= 4 False
==	Equal to	7 == 4 False
!=	Not equal to	7 != 4 True

Example:

```
if (10 == 10) {
```

```
cout << "Yes";
}
// Outputs "Yes"
```

The not equal to operator evaluates the operands, determines whether or not they are equal. If the operands are not equal, the condition is evaluated to true. For example:

```
if (10 != 10) {
cout << "Yes";
}
```

You can use relational operators to compare variables in the **if** statement. For example**:**

```
int a = 55;
int b = 33;
if (a > b) {
cout << "a is greater than b";
}
// Outputs "a is greater than b"
```

The else Statement

An if statement can be followed by an optional else statement, which executes when the condition is false.

Syntax:

```
if (condition) {
```

```
//statements
}
else {
//statements
}
```

The code above will test the condition:

- If it evaluates to true, then the code inside the if statement will be executed.
- If it evaluates to false, then the code inside the else statement will be executed.

The else Statement

For example:

```
int mark = 90;
if (mark < 50) {
cout << "You failed." << endl;
}
else {
cout << "You passed." << endl;
}
// Outputs "You passed."
```

In all previous examples only one statement was used inside the if/else statement, but you may include as many statements as you want.

For example:

```
int mark = 90;
if (mark < 50) {
cout << "You failed." << endl;
cout << "Sorry" << endl;
}
else {
cout << "Congratulations!" << endl;
cout << "You passed." << endl;
cout << "You are awesome!" << endl;
}
/* Outputs
Congrations!
You passed.
You are awesome!
*/
```

Nested if Statements

You can also include, or nest, if statements within another if statement. For example:

```
int mark = 100;
if (mark >= 50) {
cout << "You passed." << endl;
if (mark == 100) {
cout <<"Perfect!" << endl;
}
}
else {
cout << "You failed." << endl;
}

/*Outputs
You passed.
Perfect!
*/
```

Nested if else Statements

C++ provides the option of nesting an unlimited number of if/else statements. For example:

```
int age = 18;
if (age > 14) {
if(age >= 18) {
cout << "Adult";
}
else {
cout << "Teenager";
}
}
else {
if (age > 0) {
cout << "Child";
}
else {
cout << "Something's wrong";
}
}
```

The if else Statement

In if/else statements, a single statement can be included without enclosing it into curly braces.

```
int a = 10;
if (a > 4)
cout << "Yes";
else
cout << "No";
```

Loops

A loop repeatedly executes a set of statements until a particular condition is satisfied. A while loop statement repeatedly executes a target statement as long as a given condition remains true.

Syntax:

```
while (condition) {
statement(s);
}
```

The loop iterates while the condition is true.At the point when the condition becomes false, program control is shifted to the line that immediately follows the loop.

The while Loop

The loop's body is the block of statements within curly braces.

For example:

```
int num = 1;
while (num < 6) {
cout << "Number: " << num << endl;
num = num + 1;
}

/* Outputs
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
*/
```

The while Loop

The increment value can be changed. If changed, the number of times the loop is run will change, as well.

```
int num = 1;
while (num < 6) {
cout << "Number: " << num << endl;
num = num + 3;
}

/* Outputs
Number: 1
Number: 4
```

```
*/
```

Using Increment or Decrement

The increment or decrement operators can be used to change values in the loop. For example:

```
int num = 1;
while (num < 6) {
cout << "Number: " << num << endl;
num++;
}
/* Outputs
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
*/
```

Using a while Loop

A loop can be used to obtain multiple inputs from the user.

Let's create a program that allows the user to enter a number 5 times, each time storing the input in a variable.

```
int num = 1;
int number;
while (num <= 5) {
  cin >> number;
  num++;
}
```

The for loop

A for loop is a repetition control structure that allows you to efficiently write a loop that executes a specific number of times.

Syntax:

```
for ( init; condition; increment ) {
statement(s);
}
```

The init step is executed first, and does not repeat.

Next, the condition is evaluated, and the body of the loop is executed if the condition is true.

In the next step, the increment statement updates the loop control variable. Then, the loop's body repeats itself, only stopping when the condition becomes false.

For example:

```
for (int x = 1; x < 10; x++) {
// some code
}
```

The for Loop

The example below uses a for loop to print numbers from 0 to 9.

```
for (int a = 0; a < 10; a++) {
cout << a << endl;
}

/* Outputs
0
1
2
3
4
5
6
7
8
9
*/
```

In the init step, we declared a variable a and set it to equal 0.

a < 10 is the condition.

After each iteration, the a++ increment statement is executed.

The for Loop

It's possible to change the increment statement.

```
for (int a = 0; a < 50; a+=10) {
cout << a << endl;
}
/* Outputs
```

```
0
10
20
30
40
*/
```

You can also use decrement in the statement.

```
for (int a = 10; a >= 0; a -= 3) {
cout << a << endl;
}

/* Outputs
10
7
4
1
*/
```

The do...while Loop

Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop checks its condition at the bottom of the loop. A do...while loop is similar to a while loop. The one difference is that the do...while loop is guaranteed to execute at least one time.

Syntax:

```
do {
statement(s);
} while (condition);
```

while vs. do...while

If the condition evaluated to false, the statements in the do would still run once:

```
int a = 42;
do {
cout << a << endl;
a++;
}
while(a < 5);
// Outputs 42
```

The do...while Loop

As with other loops, if the condition in the loop never evaluates to false, the loop will run forever.

For example:

```
int a = 42;
do {
cout << a << endl;
} while (a > 0);
```

Multiple Conditions

Sometimes there is a need to test a variable for equality against multiple values. That can be achieved using multiple if statements.

For example:

```
int age = 42;
if (age == 16) {
cout <<"Too young";
}
if (age == 42) {
cout << "Adult";
}
if (age == 70) {
cout << "Senior";
}
```

The switch Statement

The switch statement tests a variable against a list of values, which are called cases, to determine whether it is equal to any of them.

```
switch (expression) {
case value1:
statement(s);
break;
case value2:
statement(s);
break;
...
case valueN:
```

```
statement(s);
break;
}
```

The default Case

In a switch statement, the optional default case can be used to perform a task when none of the cases is determined to be true.

Example:

```
int age = 25;
switch (age) {
case 16:
cout << "Too young";
break;
case 42:
cout << "Adult";
break;
case 70:
cout << "Senior";
break;
default:
cout << "This is the default case";
}
// Outputs "This is the default case"
```

The break Statement

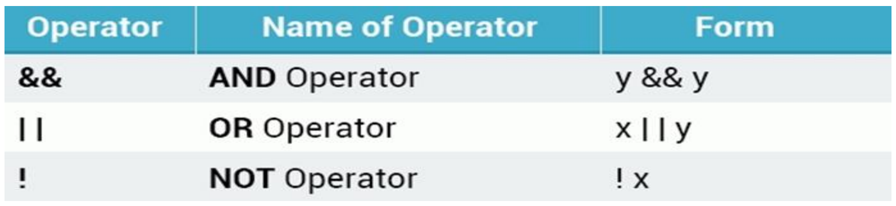
The break statement's role is to terminate the switch statement. In instances in which the variable is equal to a case, the statements that come after the case continue to execute until they encounter a break statement. In other words, leaving out a break statement results in the execution of all of the statements in the following cases, even those that don't match the expression.

For example:

```
int age = 42;
switch (age) {
case 16:
cout << "Too young" << endl;
case 42:
cout << "Adult" << endl;
case 70:
cout << "Senior" << endl;
default:
cout <<"This is the default case" << endl;
}
/* Outputs
Adult
Senior
This is the default case
*/
```

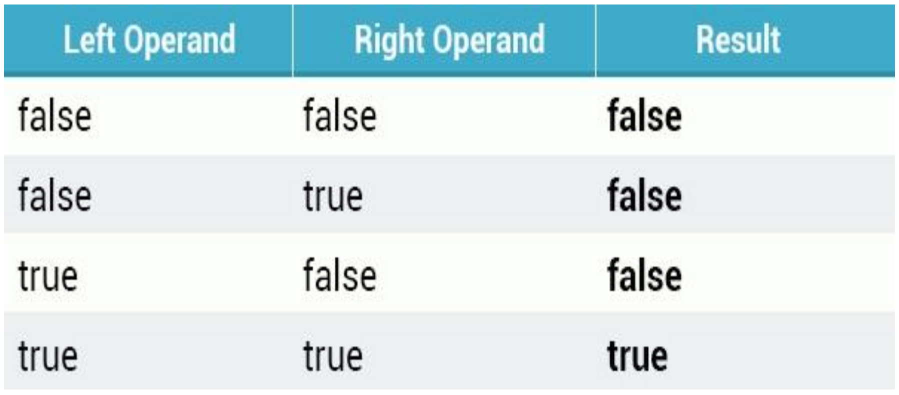
Logical Operators

Use logical operators to combine conditional statements and return true or false.



Operator	Name of Operator	Form
&&	**AND** Operator	y && y
||	**OR** Operator	x || y
!	**NOT** Operator	! x

The AND operator works the following way**:**



Left Operand	Right Operand	Result
false	false	**false**
false	true	**false**
true	false	**false**
true	true	**true**

The AND Operator

For example:

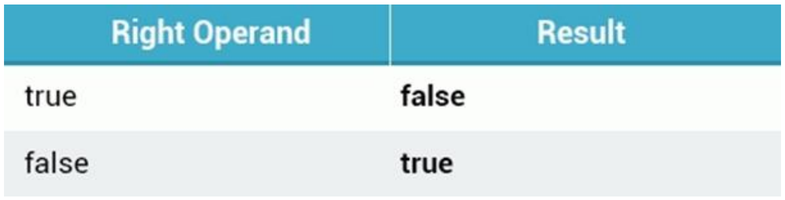
```
int age = 20;
if (age > 16 && age < 60) {
cout << "Accepted!" << endl;
}
// Outputs "Accepted"
```

Within a single if statement, logical operators can be used to combine multiple conditions.

```
int age = 20;
int grade = 80;
if (age > 16 && age < 60 && grade > 50) {
cout << "Accepted!" << endl;
}
```

Logical NOT

The logical NOT (!) operator works with just a single operand, reversing its logical state. Thus, if a condition is true, the NOT



Right Operand	Result
true	**false**
false	**true**

operator makes it false, and vice versa.

```
int age = 10;
if ( !(age > 16) ) {
cout << "Your age is less than 16" << endl;
}
// Outputs "Your age is less than 16"
```