



Basic Concepts

Welcome to 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.

Your First C++ Program

A C++ program is a collection of commands or statements.

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

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

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

Try It Yourself

Let's break down the parts of the code.

```
#include <iostream>
```

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.

Your First C++ Program

The C++ compiler **ignores** blank lines.

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;
}
```

Try It Yourself

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!";
    return 0;
}
```

Try It Yourself

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 does.

Your First C++ 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;
}
```

Try It Yourself

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;  
}
```

Try It Yourself

If the return statement is left off, the C++ compiler implicitly inserts **"return 0;"** to the end of the **main()** function. Tap **Continue** to learn more about functions, return, and other topics.

Getting the Tools

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

Reference this lesson if you need to install the software on your computer.

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.

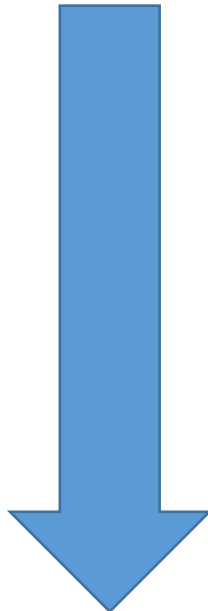
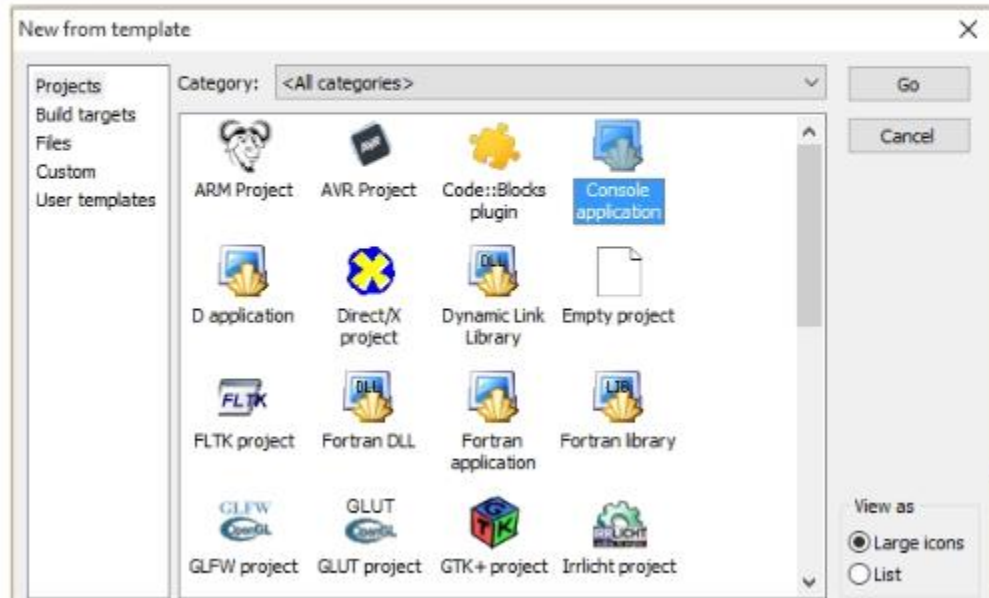
To download Code::Blocks, go to <http://www.codeblocks.org/>, Click the **Downloads** link, and choose "**Download the binary release**".

Choose your OS and download the setup file, which includes the C++ compiler (For Windows, it's the one with **mingw** in the name).

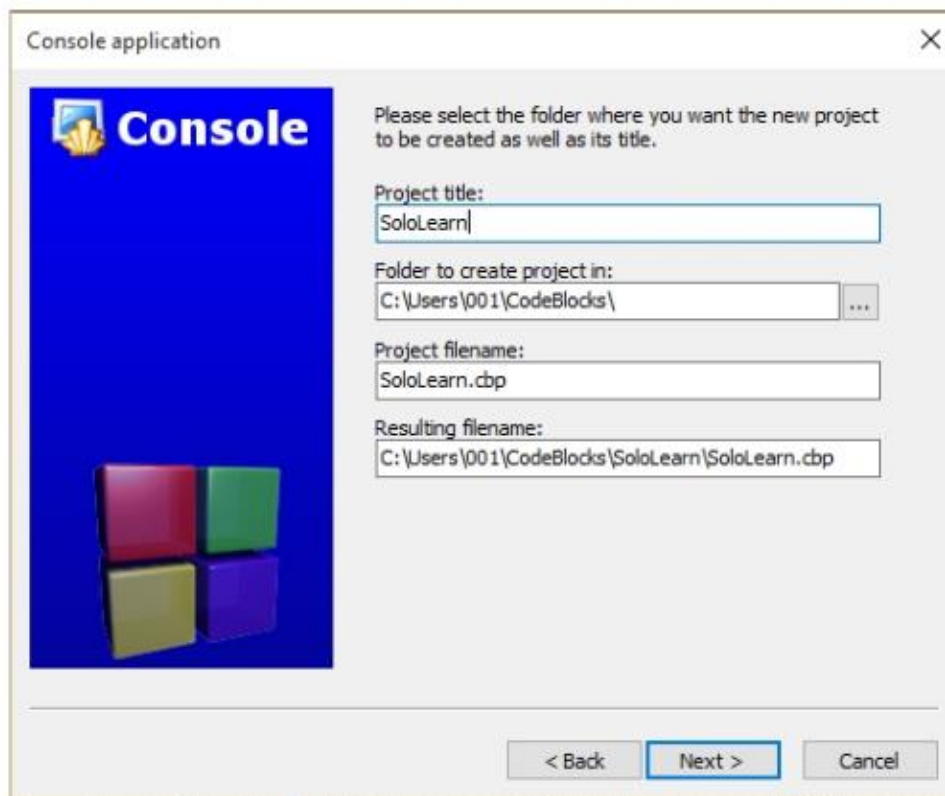
Make sure to install the version that includes the **compiler**.

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.

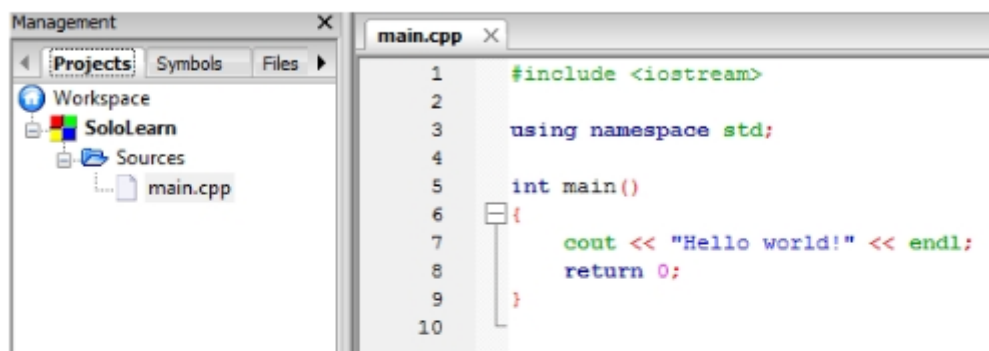


Make sure the **Compiler** is selected, and click **Finish**.

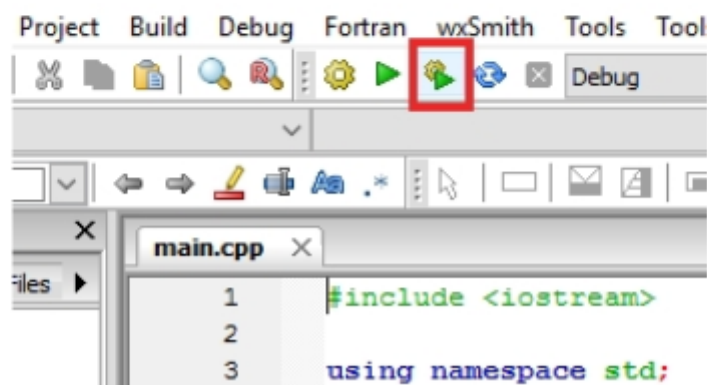
GNU GCC is one of the popular compilers available for Code::Blocks.

On the left sidebar, expand **Sources**. You'll see your project, along with its source files.

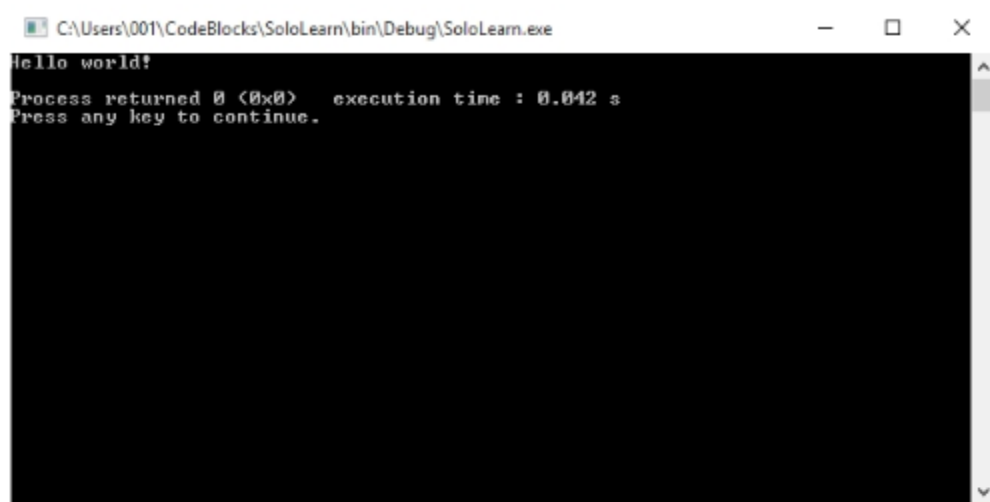
Code::Blocks automatically created a **main.cpp** file that includes a basic Hello World program (C++ source files have .cpp, .cp or .c extensions).



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



A **console window** will open and display program output.



```
C:\Users\001\CodeBlocks\SoloLearn\bin\Debug\SoloLearn.exe
Hello world!
Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

Congratulations! You just compiled and ran your first C++ program!

You can run, save, and share your C++ codes on our **Code Playground**, without installing any additional software.
Reference this lesson if you need to install the software on your computer.

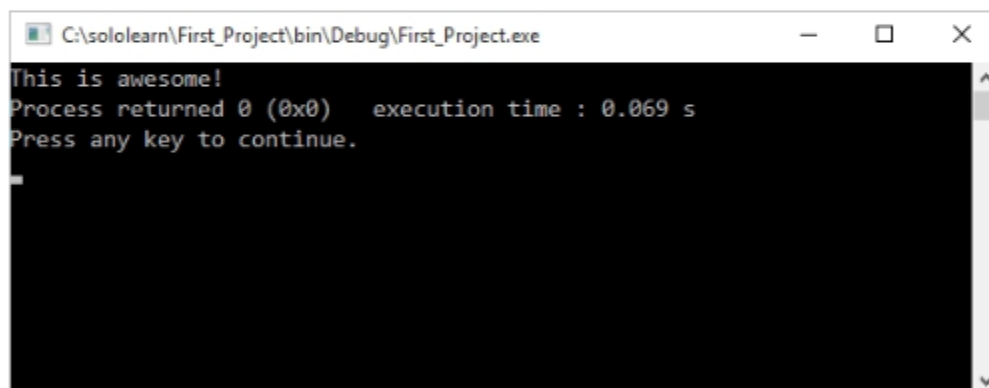
Your First C++ Program

You can add multiple insertion operators after **cout**.

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

Try It Yourself

Result:



```
C:\sololearn\First_Project\bin\Debug\First_Project.exe
This is awesome!
Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

Tap **Try It Yourself** to play around with the code!

New Line

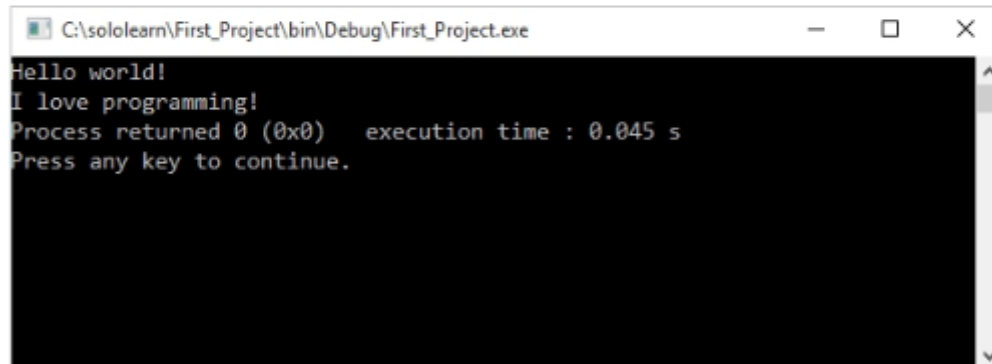
The `cout` operator does not insert a line break at the end of the output. One way to print two lines is to use the `endl` manipulator, which will put in a line break.

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

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

Try It Yourself

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



Tap Try It Yourself to play around with the code!

New Lines

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`.

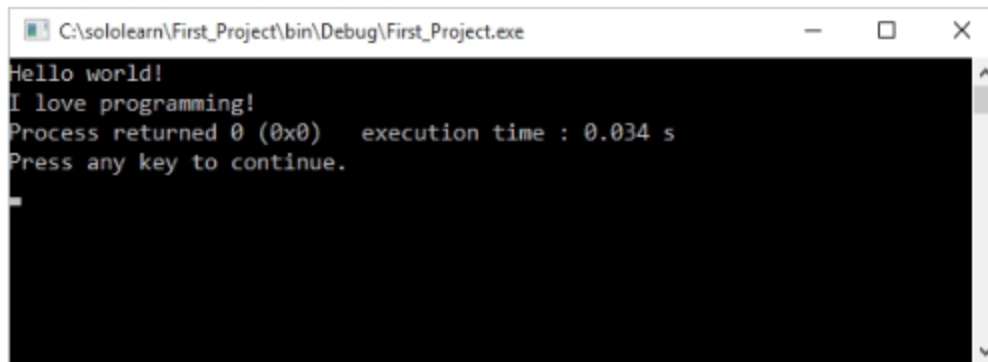
Example:

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

int main()
{
    cout << "Hello world! \n";
    cout << "I love programming!";
    return 0;
}
```

Try It Yourself

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.
```

New Lines

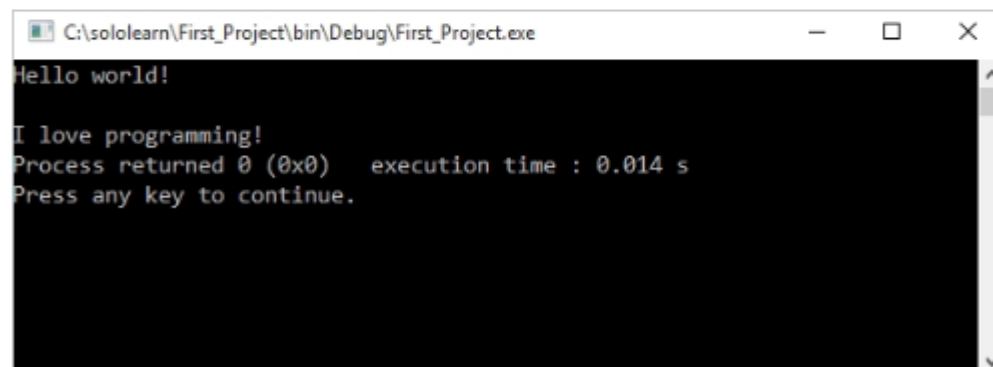
Two newline characters placed together result in a blank line.

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

int main()
{
    cout << "Hello world! \n\n";
    cout << "I love programming!";
    return 0;
}
```

Try It Yourself

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.
```

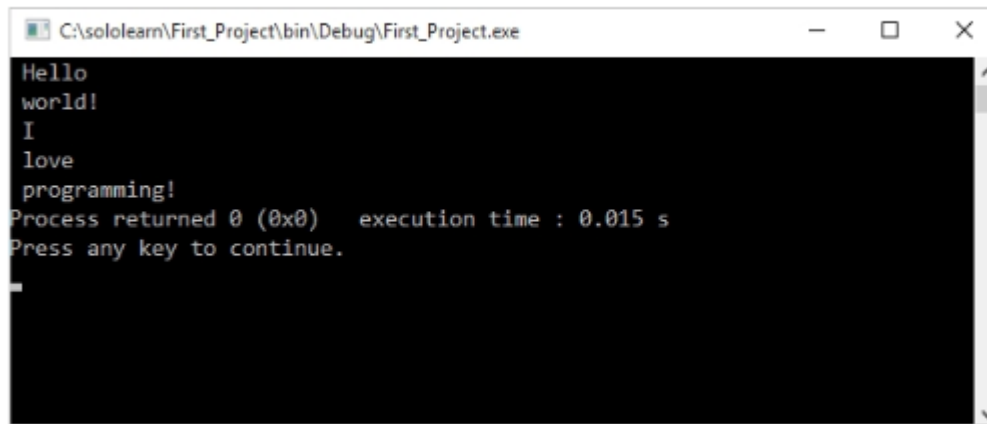
Multiple New Lines

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. The slashes tell the compiler to ignore everything that follows, until the end of the line.

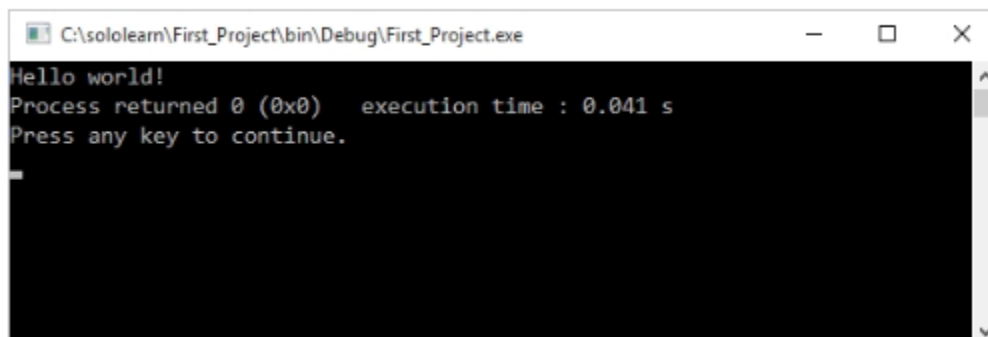
For example:

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

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

Try It Yourself

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



```
C:\sololearn\First_Project\bin\Debug\First_Project.exe
Hello world!
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

Comments make your code more readable for others.

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
   */
```

Try It Yourself

If you write a wrong code segment, don't delete it immediately. Put it into a multi-line comment, and then delete it when you find the right solution.

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!

*/
```

Try It Yourself

Adding comments to your code is a good practice. It facilitates a clear understanding of the code for you and for others who read it.

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++ offers 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;
```

Different operating systems can reserve different sizes of memory for the same data type.

Variables

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
```

Try It Yourself

The C++ programming language is **case-sensitive**, so **myVariable** and **myvariable** are two different identifiers.

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
```

Use the + operator to add two numbers.

Variables

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
```

Try It Yourself

Always keep in mind that all variables must be defined with a **name** and a **data type** before they can be used.

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;
```

You can assign a value to a variable only in a declared data type.

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;
```

As with **cout**, extractions on **cin** can be chained to request more than one input in a single statement: **cin >> a >> b;**

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;
}
```

Try It Yourself

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**.

The program will wait for as long as the user needs to type in a number.

Accepting User Input

You can accept user input multiple times throughout the program:

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

int main()
{
    int a, b;
    cout << "Enter a number \n";
    cin >> a;
    cout << "Enter another number \n";
    cin >> b;

    return 0;
}
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Accepting User Input

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;
}
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Variables

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;
```

Specifying the data type for a given variable more than once results in a syntax error.

Variables

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
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

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
```

Try It Yourself

You can use multiple arithmetic operators in one line.

Subtraction

The subtraction operator subtracts one operand from the other.

```
int x = 100 - 60;  
cout << x;  
  
//Outputs 40
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

Multiplication

The multiplication operator multiplies its operands.

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

Try It Yourself

Tap Try It Yourself to play around with the code!

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
```

Try It Yourself

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
```

Try It Yourself

Tap Try It Yourself to play around with the code!

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
```

Try It Yourself

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
```

Try It Yourself

Tap Try It Yourself to play around with the code!

Operator Precedence

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
```

Assignment operator ($=$) assigns the right side to the left side.

Assignment Operators

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

```
x *= 3; // equivalent to x = x * 3  
x /= 2; // equivalent to x = x / 2  
x %= 4; // equivalent to x = x % 4
```

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

Increment Operator

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
```

The increment operator is used to increase an **integer**'s value by one.

Increment Operator

For example:

```
int x = 11;  
x++;  
cout << x;  
  
// Outputs 12
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

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
```

The **prefix** example increments the value of x, and then assigns it to y.
The **postfix** example assigns the value of x to y, and then increments it.

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
```

The decrement operator (--) works in much the same way as the increment operator.

End.