**Practical 03**

**Getting familiar with fundamentals of programming using C++**

|  |  |
| --- | --- |
| **Objectives**   * Getting familiar with basics of C++ programming language. * Understanding and be able to write and debug simple program in C++. * Understanding the concept of variable, data types and operators in C++.   **Tools**   * Code::Blocks (Version: 13.12) or Bloodshed Dev C++ (Version: 4, 5 or higher) | |
| **Keywords:** C++, CodeBlocks, cpp, Variable, Data type, Operator. | **Duration:** 03 hours |

## Introduction

### C++ Programming Language

C++ (pronounced "see plus plus") is a general-purpose Object Oriented Programming language with high-level and low-level capabilities. It is a statically typed, free-form, multi-paradigm, usually compiled language supporting procedural programming, data abstraction, object-oriented programming, and generic programming. C++ is regarded as a mid-level language. This indicates that C++ comprises a combination of both high-level and low-level language features.

Bjarne Stroustrup developed C++ in 1979 at Bell Labs as an enhancement to the C programming language and named it "C with Classes". In 1983 it was renamed to C++. Enhancements started with the addition of classes, followed by, among other features, virtual functions, operator overloading, multiple inheritance, templates, and exception handling. The C++ programming language standard was ratified in 1998 as ISO/IEC 14882:1998, the current version of which is the 2003 version, ISO/IEC 14882:2003. A new version of the standard (known informally as C++0x) is being developed.

### History of C++

Stroustrup began work on C with Classes in 1979. The idea of creating a new language originated from Stroustrup's experience in programming for his Ph.D. thesis. Stroustrup found that Simula had features that were very helpful for large software development, but the language was too slow for practical use, while BCPL was fast but too low-level and unsuitable for large software development. When Stroustrup started working in Bell Labs, he had the problem of analyzing the UNIX kernel with respect to distributed computing. Remembering his Ph.D. experience, Stroustrup set out to enhance the C language with Simula-like features. C was chosen because it is general-purpose, fast, and portable. Besides C and Simula, some other languages which inspired him were ALGOL 68, Ada, CLU and ML. At first, the class, derived class, strong type checking, inlining, and default argument features were added to C via Cfront. The first commercial release occurred in October 1985.

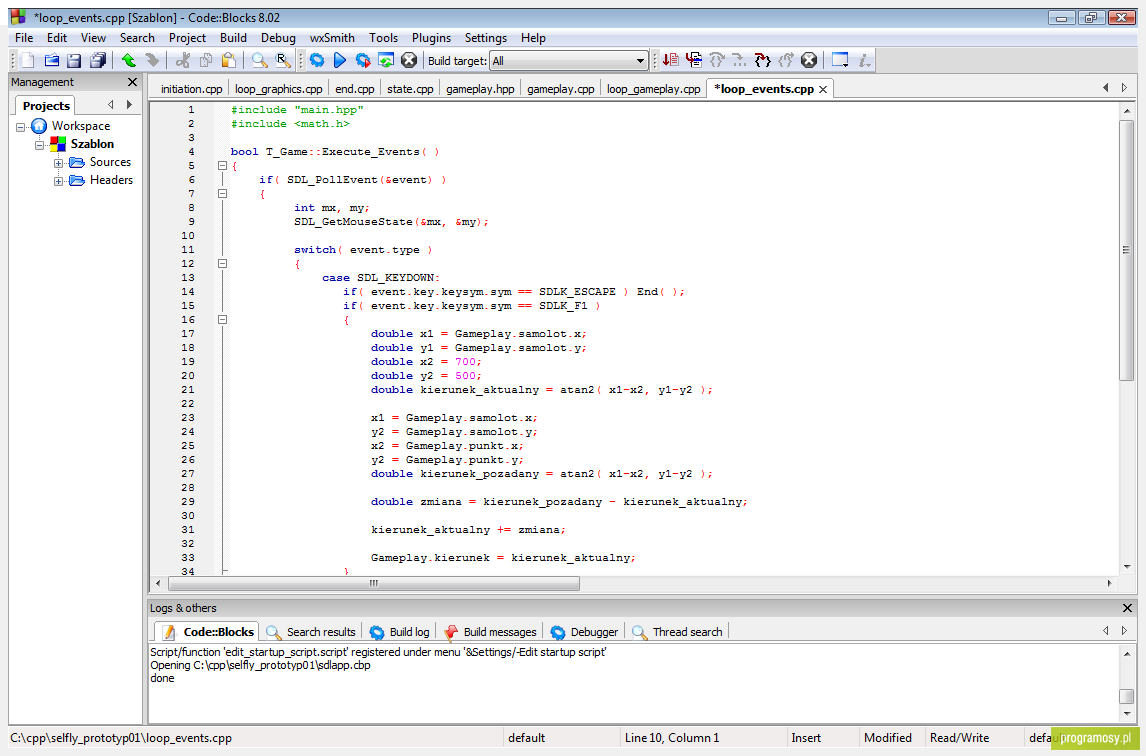
In 1983, the name of the language was changed from C with Classes to C++. New features were added including virtual functions, function name and operator overloading, references, constants, user-controlled free-store memory control, improved type checking, and BCPL style single-line comments with two forward slashes (//). In 1985, the first edition of The C++ Programming Language was released, providing an important reference to the language, as there was not yet an official standard. In 1989, Release 2.0 of C++ was released. New features included multiple inheritance, abstract classes, static member functions, const member functions, and protected members. In 1990, The Annotated C++ Reference Manual was published. This work became the basis for the future standard. Late addition of features included templates, exceptions, namespaces, new casts, and a Boolean type.

As the C++ language evolved, a standard library also evolved with it. The first addition to the C++ standard library was the stream I/O library which provided facilities to replace the traditional C functions such as printf and scanf. Later, among the most significant additions to the standard library, was the Standard Template Library.

### Code::Blocks

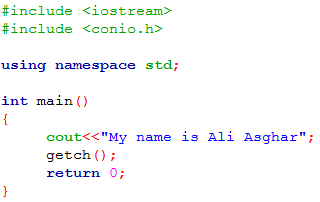
Code::Blocks is a free, open source cross-platform IDE which supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C, C++, and Fortran. It has a custom build system and optional Make support.

Code::Blocks is being developed for Windows, Linux, and Mac OS X and has been ported to FreeBSD, OpenBSD and Solaris. Statements are executed on the basis of conditions.



### Basic C++ Program

***Program (basic.cpp)***



The program starts with the two lines containing a pound sign (#) and the keyword include. These are known as preprocessor directives. The preprocessor directives are the instructions to the part of the compiler known as preprocessor which includes some extra files (codes) to the basic source program. The files iostream and conio.h are known as the header files which contain the definitions of some functions. The iostream header file contains the definition of standard input/output streams like, cout and cin where as conio.h header file includes the definitions of function getch(), getche() and others. Simply speaking the preprocessor directive #include is responsible for including the contents of the header files into the source file.

using namespace std enables a program to use all the names in any standard C++ header (such as  
<iostream>) that a program might include.

The main() is the function as the function is always along with the parentheses. The main() function is the first executable function in any C++ program. No matter where the main() function is located always the first precedence goes to the main() function and its contents. The contents of main function are enclosed in curly braces. The void before the main function says that the function main has no return type value and at the end of the function main() will not return any value.

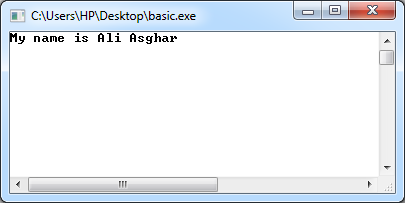
The braces “{ “and “}” also known as curly braces, enclose the block of code present in any function. “{“ is known as the opening brace and ”}” is known as closing brace. Opening brace shows the starting of the main or any function and closing brace shows the ending of the main or any function. The code of each and every function is always enclosed in the curly braces.

The cout<<”My name is Ali Asghar” tells the computer to print the string constant “My name is Ali Asghar” on the console screen. The cout is the standard output stream which directs the flow of data to the console screen. What ever written in the double quotations in cout statement is printed as it is on the console screen. Every C++ statement must end with a semicolon (also known as the statement terminator). Preprocessor directives (like #include) do not end with a semicolon.

The getch() (get character) function waits to get the character from keyboard. If you run your program without using getch() your program will show the result in just one blink and will vanish out quickly. So to make the output console screen stop in order to examine the results clearly we use getch() function. The definition of the function getch() is present in the header file “conio.h”.

When the return statement is used at the end of main, the value 0 indicates that the program has terminated successfully. According to the C++ standard, if program execution reaches the end of main without encountering a return statement, it’s assumed that the program terminated successfully—exactly as when the last statement in main is a return statement with the value 0. For that reason, we omit the return statement at the end of main in subsequent programs.

The output of basic.cpp on the console screen may look like as shown in below figure:



### Functions

Functions are the one of the fundamental building blocks of C++ language. The functions are the sub programs which contain some block of code inside the curly braces. The code enclosed inside the curly braces is known as the body of the function. The functions always start with the return type and the function name followed by the parenthesis. The parentheses tell the compiler that it is a function not a variable or any other thing. Sometimes the parentheses also contain arguments, which are the values that the function passes through its body. As the statements are the instructions to the compiler and they are just executed from the place where they are placed but if we want to execute the same block of statements at different places according to the situations of the program then the function is way which provides us the facility to execute that same block of statements (codes) at different places. The block of code present in any function can be executed at any place by calling that function at that point in the program. Any C++ program can contain number of functions or at least one i.e. main(). All the C++ programs always contain the main() function, because it is the gateway to enter in C++ and no program can run with out main() function.

In basic.cpp, the main() is only the function present. It contains the block of three line code. This three line code present in main() is always executed first in this program, because the contents of the main() functions have the highest precedence as compared to the contents of other functions.

### Statements

Statements are the instructions to the computer to work accordingly. All the statements are always terminated with the terminator “;”. This semicolon is known as terminator because it terminates the statement and tells the compiler that the statement has ended. If we don’t give the terminator at the end of any statement, the compiler will think that the statement is still going on and will mix another statement or code with it hence giving an error. In basic.cpp, there is only one statement cout<<”My name is Ali Asghar”; it just tells the computer to print the string constant present in quotation marks on the console screen. So every statement is the instruction to the computer and do remember to place the terminator at the end of each and every statement.

### Preprocessor Directives and the Header Files

The preprocessor directives are the instruction to the part of the compiler known as preprocessor. These preprocessor directives give the direction to the compiler before compiling any source code. The preprocessor directive #include is used to include the block of code of header files in to any source program without writing the whole block of code of that header file. The #include preprocessor directive provides you the facility to include thousands of lines of code of the header files into your source program. On the other hand the header files are the files with the extension .h present in the INCLUDE directory of your C++ software. These header files are the files which contain the definitions of certain functions of C++. As discussed earlier that a computer is just a dumb machine which cannot understand anything and is unfamiliar to the words like cout, cin, return, void, main etc. So in order to make the computer know what these words are we use header files, which contain the instructions to the computer that tells it how to act when it faces these kinds of codes. Now we have made the computer understand these words but the problem is this the instruction to make it understand are made in thousands of lines of codes so we cannot do such a lengthy process of writing thousands of line of codes for each tiny or long program. To overcome this major problem we can include the entire block of code present in any header file into our source file by just writing a single line statement i.e. #include <headerfilename.h>. In this way we are released from passing through such a lengthy and complex problem and we feel calm to have this feature.

The preprocessor directives are always started with the “#” (hash/pound) sign which tells the compiler that the thing that you are compiling is a preprocessor directive which is different from other. And the header files are always enclosed in “< >” angle brackets following the keyword include but in some cases it is enclosed in double quotation marks. If you enclose the header file in the angle brackets it will start the search for the header file in the standard INCLUDE directory.

#include <iostream>

But when you want to include the user defined header file the header file is enclosed in quotation marks.

#include “ali.h”

The quotation marks tells the compiler to start the search for the header file in the current directory—the directory usually containing the source files.

In basic.cpp, there are two header files included in the source program. One is iostream which contains the definitions of standard input and output streams like, cout and cin. The other header file is conio.h which contains the basic definition of the functions like, getch(), getche(), etc.

### Standard Input and Output Streams

The term stream means the flow of the data. The data flown towards the system through the input devices is known as input stream whereas the data flown from the system to some output devices is known as output stream. The standard input stream is flown from the keyboard to the system and the standard output stream is flown from the system to the console screen. In order to control this flow of data to and from the system there are two statements used. The first “cout” which controls the standard output stream and the other “cin” which controls the standard input stream. The definitions of both the keywords is written in the header file iostream (input/output stream), so to get access to these statements you must first include the header file iostream in your source program.

### Outputting with cout

The identifier cout is actually an object which is predefined in C++. The cout statement is responsible for displaying the text, numbers and some special characters and graphical symbols on the console screen. The simplest structure of cout statement is given as:

cout<<”My name is Ali Asghar”;

In above syntax the keyword cout tells the compiler that it is a standard output stream and treat it as accordingly. The text in the double quotation marks ”My name is Ali Asghar” is the string constant and is printed on the console screen as it is. The operator << is known as insertion or put to operator. It directs the contents of the variable on its right to the object on its left. The insertion operator can be used multiple times in single cout statement and it is perfectly legal as in below statement:

cout<<”My name is “<<”Ali Asghar”;

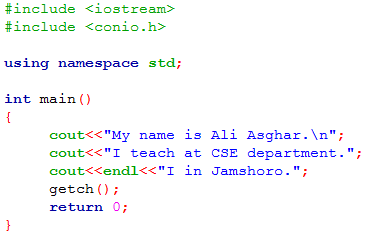
Except printing the string constants, the situation also come in which we have to print out the values of certain variables on the console screen then we can use the variable name in cout statement to print its value on the console screen as in below statement:

cout<<”The addition of two numbers is: “<<Add;

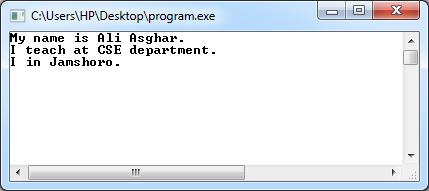
To make the variable Add effective in cout statement it should be declared and defined properly in the program. You can also print the values of several variables in using cout statement as in below statement, which prints the number of apples and oranges in the box.

cout <<”There are “<<a<<”apples”<<”and “<<b<<”oranges in the box”;

As the cout is a statement therefore each and every cout statement should be terminated with the terminator “;”. Let’s examine the following program:



The above program’s code contain three cout statements. The first cout statement prints the string constant “My name is Ali Asghar.” on the screen as it is where as “\n” is the escape sequence which moves the cursor to the new line so that the coming line will be printed on the fresh line. Then the next cout statement flows the string constant ”I teach at CSE department.”; to the console screen on the new line. In the last cout statement we face a new unfamiliar word endl, it is known as the manipulator. The manipulators are the instructions to the output stream which modify or manipulate the way the data is displayed in the console screen. The manipulator endl has the same effect of causing a linefeed as the \n has but it is somewhat clearer. The endl manipulator also causes the buffer to be flushed. So, on conclusion the third cout statement in SOS.cpp, flows the string constant ”I live in Jamshoro.” to the console screen on new line.

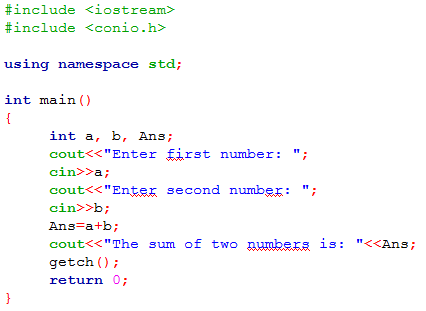


### Inputting with cin

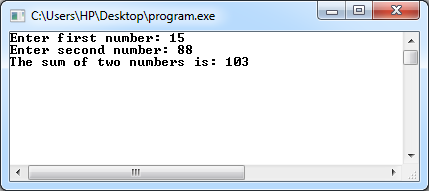
The identifier cin is an object which is predefined in C++. The cin statement is responsible for getting input and passing for processing during the run-time. The simplest structure of cin statement is:

cin>>number;

In above syntax the keyword cin tells the compiler that it is a standard input stream and treat it as accordingly. The operator >> is known as extraction or get from operator. It takes the value from stream object on its left and places it in the variable on its right. Let’s examine the following program



The above program first initializes that the variables a, b and Ans contain the integer data type values. The first cout statement sends the message to enter first number from your keyboard and when you enter any number, the cin statement places that number in the variable a. Then the program gives another message to enter second number and when you enter any number that will be placed in the variable b. The statement Ans=a+b then adds the two numbers and places the result in the variable Ans. Finally the last cout statement shows the addition of both numbers.



### Comments

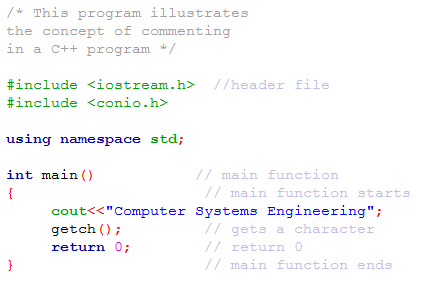
Commenting is very effective feature in any programming language as it makes the source program user friendly up to some extent. Comments are un-executable lines in any source program. Comments make the program more readable and understandable to the user or even an unknown reader. They help the programmer to make him remember that for this purpose you have written this portion of code. And also if a person is unfamiliar with the program, the comments will help him by telling him what is happening in a certain line or portion of the program. When you write a program and suppose you are not in touch with it for a two or three months so when you come back to your program after such a long period you will forget that for what I had written it and what it tell? So if you had given the comments in your program you would not be in such a great trouble.

In small programs it is not a huge problem to forget some code lines but in case of working on some programs having hundreds or thousands of code line or working on a project you have to deal with several different blocks of codes so, in these situations there are a lot of chances to forget the construction of your own program. So here the comments play their key role.

There are two ways to comment in C++ source file. The first way it to start comment with the two back slashes “//” , this causes the compiler to consider the whole line as a comment. The other way is to start the comment with a back slash and an asterisk “/\*” and terminate the comment with an asterisk and back slash “\*/”, which cause the certain block of line as a comment.

The first way only make a single line as a comment but if we want to make the 100 lines as the comment then we can not use the double backslashes “//” hundred of times. But at this situation the second way suites very well in which we will place “/\*” the starting of the comment and “\*/” at the ending of the comment.

Examine the following program:



### Variables

The basic definition of the variable says that the variables are those memory locations whose value can be varied/altered according to the particular situation. Like in other programming languages variables are one of the major building blocks of C++ programming language. The variables set the location into the memory and give it certain name so you can store certain value and access the particular location of memory. The name given to the variable is known as identifier. It is so called because it identifies/indicates certain memory location. In C++ programming language there are certain rules for identifiers so, being in the boundary of those you can declare an identifier.

Some of the rules are given as:

1. The identifier can contain letters form a-z, numbers form 0-9 and an underscore sign.
2. The identifier can be in upper or lower case but the variable in upper case will differ from the variable in lower i.e. ANS is not same as ans or Ans.
3. The first character of the identifier must be letter or an underscore sign.
4. The identifier should not contain any space (white space) within it.
5. You can also give underscore sign in the middle of the identifier as an space for your ease for example, square\_inch.
6. The identifier must not be same as the keywords—the words predefined in C++ which have their own specific meaning and function. Like, main, void, return, cout, cin etc. are keywords so the identifier should not be like them.
7. The identifier can be as long as you like, but only the first 247 characters (in Visual C++) or 250 characters (in C++ Builder) will be recognized.
8. The identifier must be unique through out the program i.e. if you have declared the identifier Var1 so, to access or call it you must give its same name Var1.

*Some valid identifiers are:* Var, var, VAR, Var1, VAR1, Var\_one, \_Var1, \_Var\_one\_of\_one etc.

*Some of invalid identifiers are:* 1Var, 1\_var, void, cout, etc.

***Hint:*** The standard and easy way to declare the identifier is to declare all characters of its in lower case like, var1, var2 etc.

#### Declaring and Defining a Variable

You might think that the two words declaring and defining are equivalent but in fact there is a lot of difference between both of them. The declaring is the process of giving a name to the variable and its data type. The data type means that which type of value will be stored in that variable whereas the name of the variable must follow the identifier rules of C++.

int var1;

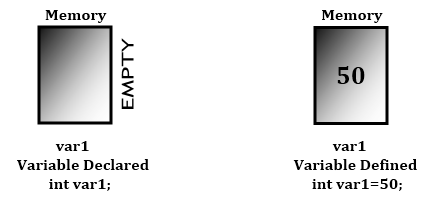
The above line is an example of declaring a variable in which the variable is given the name var1 and integer data type declared for it which tells the computer that the value stored in the variable var1 must be an integer.

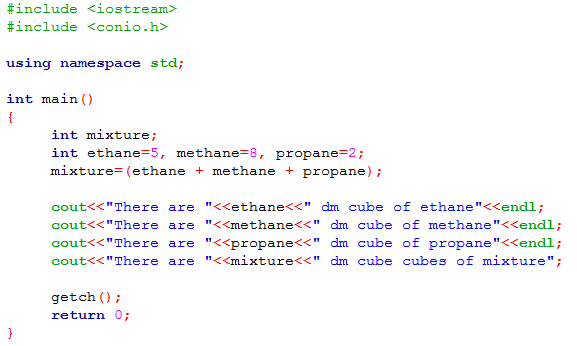
Whereas the process of initializing certain value to the variable at the time of declaration is referred as defining a variable. The line below illustrated the concept of defining a variable:

int var1=50;

In above line the variable var1 is initialized with the integer value 50.

In defining a variable we set/initialize some values to the variable before the compiling.





The above program starts with two preprocessor directives which include two header files which contain definitions of some functions and keywords as, cout and getch(). Then in main function one variable mixture is declared and three variables ethane, methane and propane are defined. The first variable mixture is only declared as it is only given the name and its data type and no value is set in the memory location of the variable. Where as in the second variable initialization statement the memory location is set for the three variables and a certain value is initialized to the memory location of the variables. The last four cout statements prints the values of the variable stored in to them on the console screen with some strings.

### Data Types

The variables are the fundamental building blocks of C++ as discussed earlier, which set the memory location by giving them certain names. The variables can store certain data/values into those memory locations but the data/values stored in to the variables are of different types and occupy different sizes of memories.

#### Integer data type

The numeric data having no fractional/decimal part is known as integer data. The integer data type variables can only store and represent integer data. The integer data type is of three types i.e. type int, type short and type long.

Type int occupies 4 bytes (32 bits) of memory. In type int you can store the integers with in the rage of -2,147,483,648 to 2,147,483,647. To define or declare a type int variable use the keyword int before the variable name like,

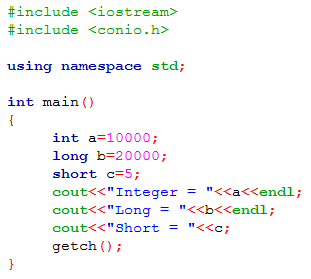
int var1= 1000000000;

Type short occupies 2 bytes (16 bits) of memory. In type short you can store the integers within the range of -32,768 to 32,767. To define or declare a type short variable use the keyword short before the variable name like,

short var1=10;

Type long occupies 4 bytes (32 bits) of memory. In type long you can store the integers with in the rage of -2,147,483,648 to 2,147,483,647. To define or declare a type long variable use the keyword long before the variable name and place the letter L after the integer constant like,

long var1=1000000000L;



#### Floating point data type

The numeric data having fractional/decimal part is known as floating point data. The floating point data can store and represent floating point data. The floating point data type is of three types i.e. type float, type double and type long double.

Type float occupies 4 bytes (32 bits) of memory. In type float you can store the floating point values within the range of 3.8 x 10-38 to 3.8 x 1038 with the precision of seven (7) digits. To define or declare a type float variable use the keyword float before the variable name and place the letter F after the floating point constant like,

float PI=3.1415F; but placing F is optional.

The other two floating point data types are same as type float but they offer wider range of values and precisions.

Type double occupies 8 bytes (64 bits) of memory. In type double you can store the floating point values within the range of 1.7 x 10-308 to 1.7 x 10308 with the precision of fifteen (15) digits. To define or declare a type double variable use the keyword double before the variable name like,

double PI=3.141592654;

Type long double occupies 10 bytes (80 bits) of memory. In type long double you can store the floating point values within the range of 1.2 x 10-4932 to 1.2 x 104932 with the precision of nineteen (19) digits. To define or declare a type long double variable use the keyword long double before the variable name and place the letter L after the floating point constant like,

long double PI=3.141592654546845348645454L; but placing L is optional.

In floating point data types you can write the floating point constants using exponential notation, which is the way to write very large or very small numbers in the power of ten. Like instead of writing 2,000,000,0 you can write 1.0E7 in exponential notation. Similarly for 52347.2 you can write 5.2E4 and for 0.0000006024 you can write 6.02E-7.

double atoms=6.02E-7;

If you want to define a floating point variable then in type float place a letter F in the end of the constant number, in type double you don’t have to identify the complier that it is a constant value it considers it as default but in type long double you have to place the letter L after the constant number. But placing F and L is optional.

float PI=3.1415F;

double PI=3.141592654;

long double PI=3.141592654546845348645454L;

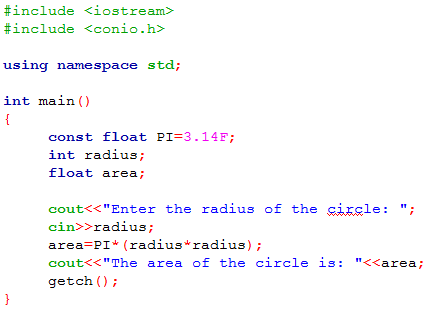
You can also make the value of the floating point variable constant through out the function by using the constant qualifier i.e. const. To define a constant floating point variable place the keyword const before the data type of the variable as in:

const float PI=3.1415F;

const double PI=3.141592654;

const long double PI=3.141592654546845348645454L;

Examine the following program:



This program defines the constant value of the variable PI and declares the integer variable radius and a type float variable area. The program gets the value of radius during the run-time in integer type and then puts it in the expression area=PI\*(radius\*radius) and calculates the value of area and finally shows the value of area.

#### Character Data Type

The data containing individual characters is known as character data. The character data type variables can only store and represent the characters. The character data type variables can only store single character at a time in a single variable.

Type char occupies 1byte (eight bits) of the memory. In type char you can store the characters within the range of integers -128 to 127, whereas the integers -128 to 127 represent the ASCII equivalents to the characters. To declare a character data type use keyword char before the variable name as in,

char ch;

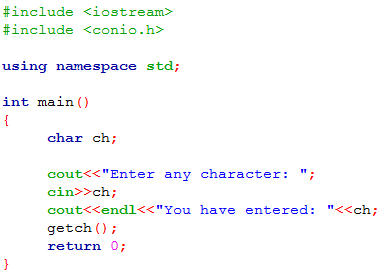
The above declaration indicates that the variable ch is a character type data variable and in the program it will only store the characters. And to define a variable you can either give the character in single quotation marks or you can give the ASCII equivalent to that character. For example if we want to store the character A (capital A letter) in the variable ch then we can write as in,

char ch=’A’;

char ch=65;

In first line the character A is enclosed in single quotation marks so the character A will be store in the variable ch where as in second line the number 65 is the ASCII equivalent to the character A so the computer will translate it in to character A and will store it in the variable ch. No matter which method you perform the aim of both the methods is same.

Examine the following program:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Numerical Range** | | **Precision** | **Bytes of Memory** |
| **Low** | **High** |
| char | -128 | 127 | n/a | 1 |
| short | -32,768 | 32,767 | n/a | 2 |
| int | -2,147,483,648 | 2,147,483,647 | n/a | 4 |
| long | -2,147,483,648 | 2,147,483,647 | n/a | 4 |
| float | 3.4 x 10-38 | 3.4 x 1038 | 7 | 4 |
| double | 1.7 x 10-308 | 1.7 x 10308 | 15 | 8 |
| long double | 1.2 x 10-4932 | 1.2 x 104932 | 19 | 10 |

#### Unsigned data types

The data types int, short, long and char have their range with in which they can store certain values and these ranges start from some negative number to positive number. So by eliminating negative numbers we can extend the size of the data types. Doing this will make the data types to store large values as twice as the signed data types do.

The unsigned data types are used when we are dealing with only positive numbers. To convert a signed data type into an unsigned data type place the keyword unsigned before the data type as in,

unsigned int var1=12000;

Type unsigned char occupies 1 byte (8 bits) of memory. In type unsigned char you can store the characters within the range of integers 0 to 255, whereas the integers 0 to 255 represent the ASCII equivalents to the characters. To declare an unsigned data type variable use keywords unsigned char before the variable name as in,

unsigned char ch=250;

Type unsigned short occupies 2 bytes (16 bits) of memory. In type unsigned short you can store the integers within the range of 0 to 65,535. To declare an unsigned short data type variable use the keywords unsigned short before the variable name as in,

unsigned short var1=50;

Type unsigned int and type unsigned long occupy 4 bytes (32 bits) of memory. In both the data types you can store the integers within the range of 0 to 4,294,967,295. To declare an unsigned int and unsigned long data type use the keywords unsigned int and unsigned long respectively before the variable names as in,

unsigned int var1=10000;

unsigned long var1=10000;

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Numerical Range** | | **Bytes of Memory** |
| **Low** | **High** |
| char | 0 | 255 | 1 |
| short | 0 | 65,535 | 2 |
| int | 0 | 4,294,967,295 | 4 |
| long | 0 | 4,294,967,295 | 4 |

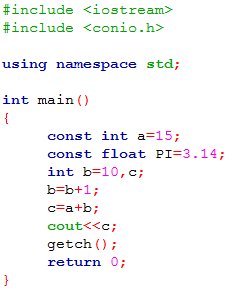
### The const Qualifier

The keyword const is known as the constant qualifier. It specifies that the value of the variable will remain constant and will not be altered throughout the function. If any attempt is made to alter the value of the variable the compiler will give the error. It is always placed before the data type of the variable as in,

const float PI=3.1415F;

It specifies that the variable PI stores the floating point number 3.1415 and this value will not be altered and will remain constant.

Examine the following program:



### Assignment Statements

The statements that assign certain values to certain variables are known as assignment statement. The assignment statements always contain an equal sign which is actually the assigner of the values to the variables. Like other statements the assignment statements always terminate with the terminator. The statements

var1=10;

var1=var1+15

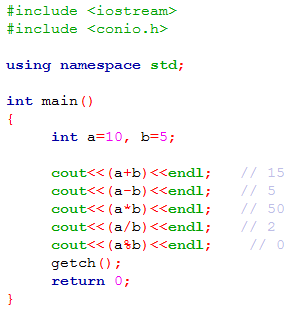
var1=var2+15;

are the assignments statements which assign the values to the variable var1. The assignment statements assign the value from the right of the = sign to the variable on the left. To assign a variable it must be noted that the variable must be declared before it is assigned.

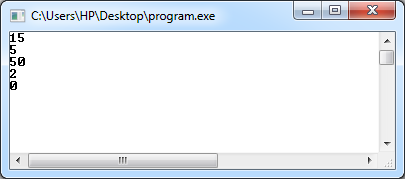
### The endl Manipulator

Manipulators are the instructions to the output stream that modify the output in various ways. The endl manipulator is also the instruction to the output stream i.e. cout, which manipulates/modifies the output displayed in the console screen. The endl manipulator used in output stream cout causes the line feed. Whenever the endl manipulator comes in cout statement it moves the cursor to the starting of the next line. It has same effect as \n escape sequence has to cause the line feed but endl is somewhat clearer and it also causes the output buffer to be flushed. The endl causes the next string constant in the cout statement to be printed on the new fresh line.

Examine the following program:



In the first cout statement the variables a and b are added, in second the variables are subtracted, in third the variables are multiplied, in fourth the variables are divided and in fifth the variable are remaindered. The output of this program is:



### Arithmetic Assignment Operators

The operators used in assignment statements to shorten and clarify the code are known as arithmetic assignment operators. They are +=, -=, \*=, /= and %=. In some situation probably you would have faced some statements like,

a=a+b;

b=b-10;

var1=var1\*50;

var1=var1/var2;

In these statements a certain value is given to a variable by performing some arithmetic operation but one variable is used twice in the statements which is really stretching therefore C++ also offer other approach to shorten and make the assignment statements more clearer. In this approach you can use the arithmetic assignment operators +=, -=, \*=, /= and %= which have the same effects as above.

In C++ a=a+1; and a+=1; have the same effects. Some other equivalents are,

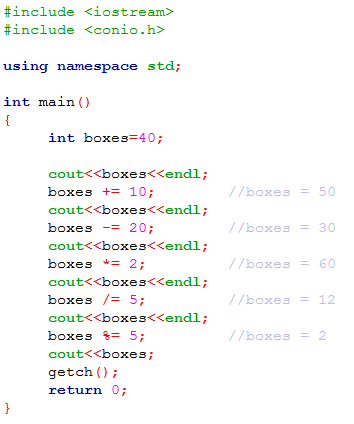
electrons += 1; is same as electrons=electrons+1;

electrons -= 1; is same as electrons=electrons-1;

electrons \*= 1; is same as electrons=electrons\*1;

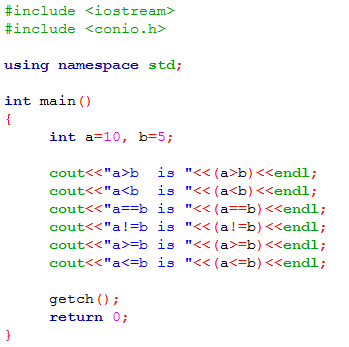
electrons /= 1; is same as electrons=electrons/1;

electrons %= 1; is same as electrons=electrons%1;



### Relational Operators

The relational operators are the operators which compares the operands. There are only two possible values that the relational operators can give after performing the operation on the operands i.e. 0 or 1. If the condition becomes true when the operands are compared with relational operators we get the value 1. On the other hand if the condition becomes false when the operands are compared with relational operators we get the value 0. There are six relational operators they are, Greater than >, Less than <, Equal to =, Not equal to !=, Greater than or equal to >=, Less than or equal to <=.



|  |  |
| --- | --- |
| **Operator** | **Description** |
| > | Greater than |
| < | Less than |
| == | Equal to |
| != | Not equal to |
| >= | Greater than or equal to |
| <= | Less than or equal to |

### Logical Operators

Logical operators logically combine the Boolean variables (0 and 1). There are three logical operators And &&, OR || and Not !. They have wide scope when used with relational operators. They combine the Boolean value obtained by relational operators. Like in relational operators we can only compare two operands and get the true or false value of only single comparison but if we want to make the condition satisfied only when the two or more comparison are satisfied then we use logical operators.

And && operator combines the Boolean values and makes the condition satisfied only when all the comparisons are satisfied as,

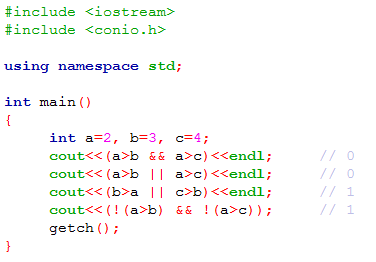
(a>b && a>c && a>d)

OR || operator combines the Boolean values and makes the condition satisfied when atleast one comparison becomes true as,

(a>b || a>c || a>d)

Not ! operator combines the Boolean values and makes the condition satisfied only when all the comparisons become false as,

(!(a>b)&& !(a>c)&& !(a>d))



|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | Logical AND |
| || | Logical OR |
| ! | Logical NOT |

### Increment Operators

The increment operators in C++ can increment or decrement the value of the variables by one. The increment operators lie in the category of unary operates. In C++ unary operators have the higher precedence than the other operators. To increment the variable by the value of one the operator ++ is used and to decrement the variable by the value of one the operator -- is used. The operators ++ and -- can be used in the two ways as a prefix or as a postfix. When the operator is used in the prefix, the variable follows the operator and when the operator is used in the postfix the operator follows the variable. Look at the below codes:

int ans=10;

cout<<++ans; //operator used as prefix

int ans=10;

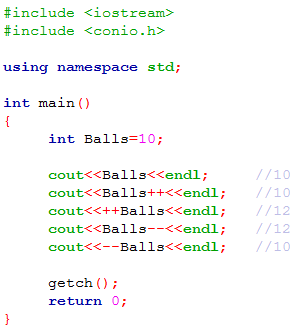
cout<<and++; //operator used as postfix

In first two line codes the value of the variable “ans” is incremented first and then it is printed on the console screen but in the second two line codes the value of the variable “ans” is first printed on the console screen and then it is incremented the one. Operators used as prefix and as postfix behave differently.

In C++,

Word=Word+1 is same as Word += 1 is same as ++Word

Word=Word-1 is same as Word -= 1 is same as --Word



## Procedure

### Problem example

Consider the following problem example. We are going to write C++ source code in Code::Blocks IDE.

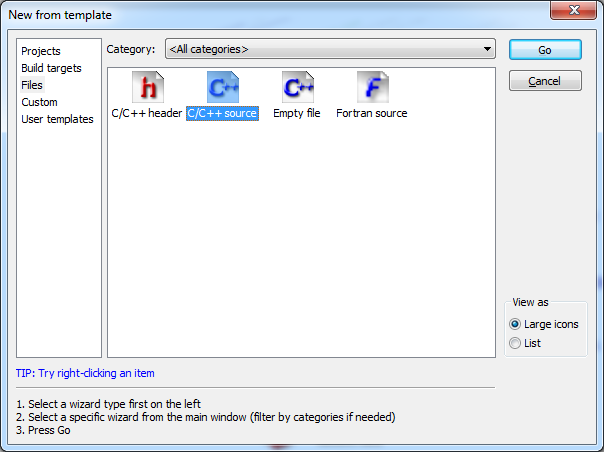
***Problem statement:*** Write a program in C++ that accepts the width and the height of a rectangle from the user and prints the area of the rectangle.

### Source Code in Code::Blocks

**Step 01:** Run Code::Blocks IDE.

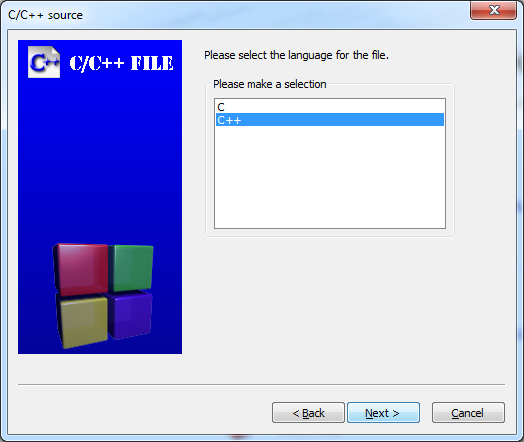
**Step 02:** Create new cpp file by going in to *File Menu>New>File.*

**Step 03:** Select C/C++ Source from template dialog box.

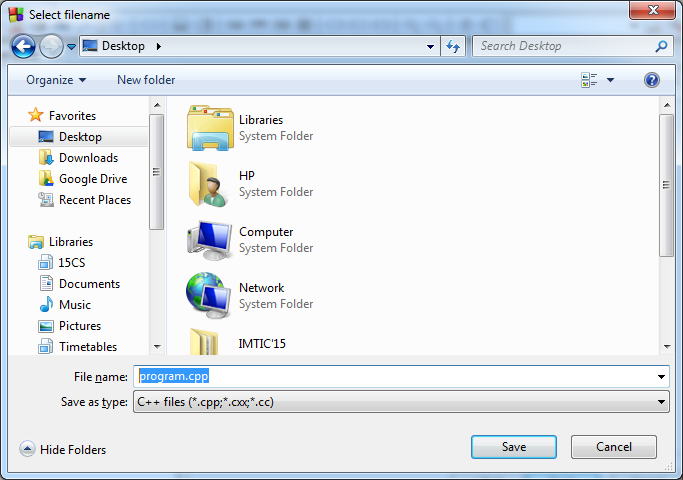


**Step 04:** Press *Go* button and then press *Next* button.

**Step 05:** Select *C++* and press *Next*.

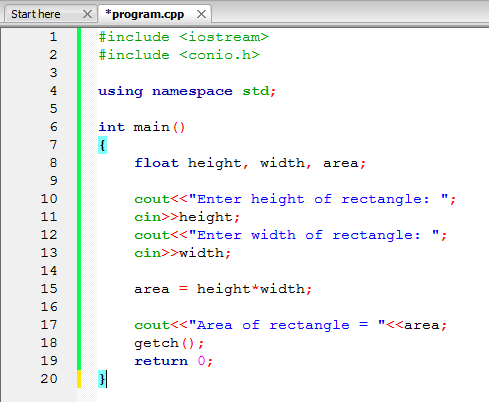


**Step 06:** Press  button and create a new file with .cpp extension by choosing the destination location of the file.



**Step 07:** Click *Finish* button.

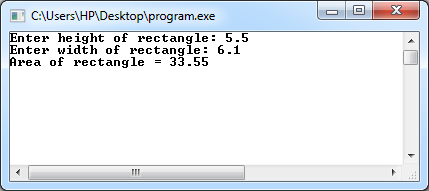
**Step 08:** Write your code in Code Window.



**Step 09:** Click button to compile the code.

**Step 10:** Debug any error in the code and recompile it.

**Step 11:** Click button to run the code.

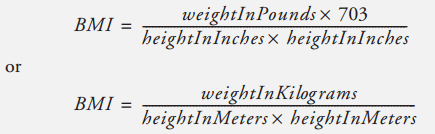


EXERCISE

1. Write a single C++ statement to accomplish each ofthe following (assume that using directives have not been used):
2. Declare the variables c, thisIsAVariable, q76354 and number to be of type int.
3. Prompt the user to enter an integer. End your prompting message with a colon (: ) followed by a space and leave the cursor positioned after the space.
4. Read an integer from the user at the keyboard and store it in integer variable age.
5. Print the message "This is a C++ program" on one line.
6. Print the message "This is a C++ program" on two lines. End the first line with C++.
7. Print the message "This is a C++ program" with each word on a separate line.
8. Print the message "This is a C++ program". Separate each word from the next by a tab.
9. Write a statement (or comment) to accomplish each of the following (assume that using directives have been used for cin, cout and endl ):
10. State that a program calculates the product of three integers.
11. Declare the variables x, y, z and result to be of type int (in separate statements).
12. Prompt the user to enter three integers.
13. Read three integers from the keyboard and store them in the variables x, y and z.
14. Compute the product of the three integers contained in variables x, y and z, and assign the result to the variable result.
15. Print "The product is " followed by the value of the variable result.
16. Return a value from main indicating that the program terminated successfully.
17. Using the statements you wrote in Exercise 2, write a complete program that calculates and displays the product of three integers. Add comments to the code where appropriate. [Note: You’ll need to write the necessary using directives.]
18. What, if anything, prints when each of the following C++ statements is performed? If nothing prints, then answer “nothing.” Assume x = 2 and y = 3.
19. cout << x;
20. cout << x + x;
21. cout << "x=";
22. cout << "x = " << x;
23. cout << x + y << " = " << y + x;
24. z = x + y;
25. cin >> x >> y;
26. // cout << "x + y = " << x + y;
27. cout << "\n";
28. Write a program that asks the user to enter two numbers, obtains the two numbers from the user and prints the sum, product, difference, and quotient of the two numbers.
29. Write a program that prints the numbers 1 to 4 on the same line with each pair of adjacent numbers separated by one space. Do this several ways:
30. Using one statement with one stream insertion operator.
31. Using one statement with four stream insertion operators.
32. Using four statements.
33. Write a program that reads in the radius of a circle as an integer and prints the circle’s diameter, circumference and area. Use the constant value 3.14159 for π. Do all calculations in output statements.
34. Write a program that prints a box, an oval, an arrow and a diamond as follows:



1. The formulas for calculating Body Mass Index (BMI) are



Create a BMI calculator application in C++ that reads the user’s weight in pounds and height in inches (or, if you prefer, the user’s weight in kilograms and height in meters), then calculates and displays the user’s body mass index.