[Pick the date]

CMP1O113

INTRODUCTION TO COMPUTING

University of Lahore | Designed by Zahid muneer

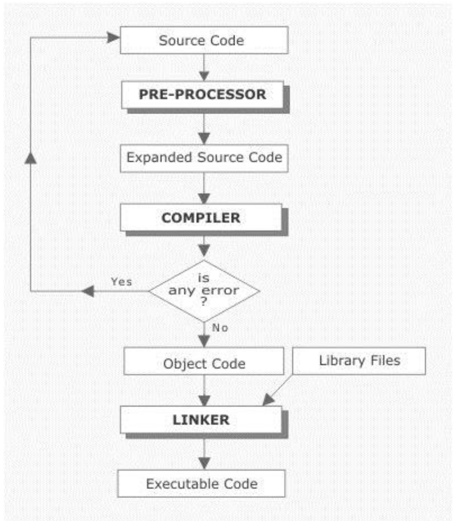
LAB MANUAL : 2
Introduction to Computing

Lab Session 9: Introduction to Dev. C++, Basic Structure & I/O Commands

Objective: Introduction to C++ Compilerand Installation of Dev C++. And to get acknowledged about the basic C++ Program structure and basic commands for input and output.

C++ Compiler:

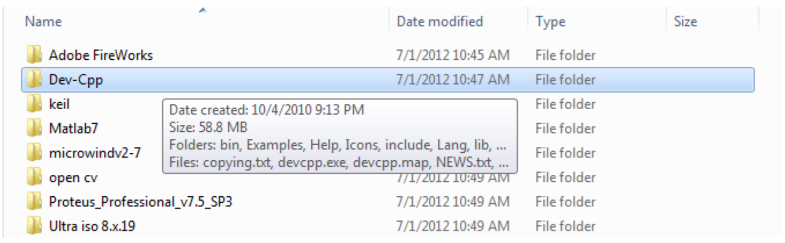
A C++ compiler is itself a computer program that's only job is to convert the C++ program from our form to a form the computer can read and execute. The original C++ program is called the **"source code"**, and the resulting compiled code produced by the compiler is usually called an **"object file"**. Before compilation the **preprocessor** performs preliminary operations on C++ source files. Preprocessed form of the source code is sent to compiler. After compilation stage object files are combined with predefined libraries by a linker, to produce the final complete file that can be executed by the computer. A library is a collection of pre-compiled "object code" that provides operations that are done repeatedly by many computer programs.



Compilation and Linking of a C++ Program

Installation steps of Dev C++ compiler:

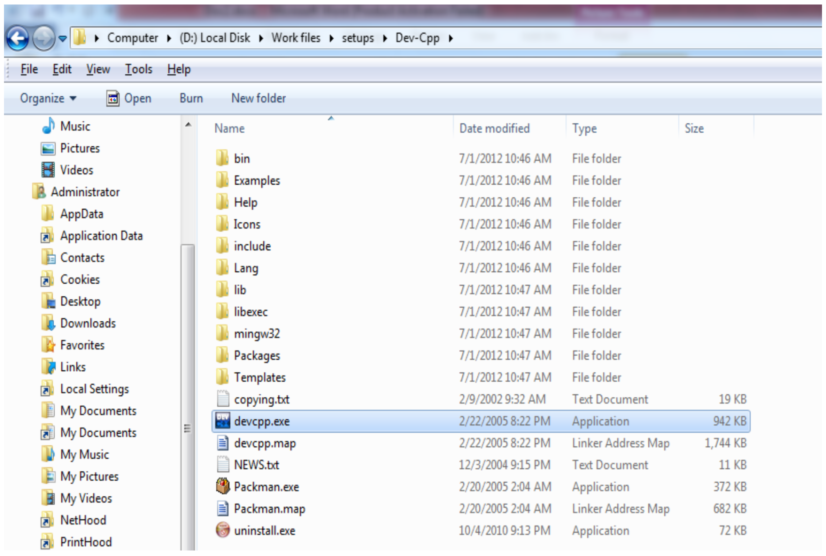
Step 1: Download the setup and Unzip the folder. Click on the folder Dev C++



Name	Date modified	Type	Size
Adobe FireWorks	7/1/2012 10:45 AM	File folder	
Dev-Cpp	7/1/2012 10:47 AM	File folder	
keil		File folder	
Matlab7		File folder	
microwindv2-7		File folder	
open cv	7/1/2012 10:49 AM	File folder	
Proteus_Professional_v7.5_SP3	7/1/2012 10:49 AM	File folder	
Ultra iso 8.x.19	7/1/2012 10:49 AM	File folder	

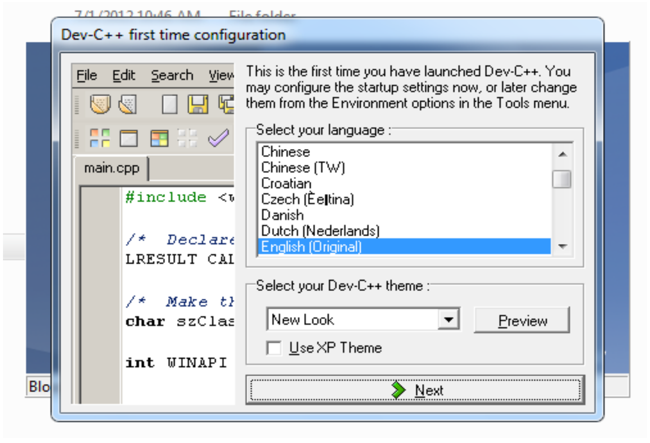
Date created: 10/4/2010 9:13 PM
Size: 58.8 MB
Folders: bin, Examples, Help, Icons, include, Lang, lib, ...
Files: copying.txt, devcpp.exe, devcpp.map, NEWS.txt, ...

After you open the folder, you will find the following files. Double click devcpp.exe file



Name	Date modified	Type	Size
bin	7/1/2012 10:46 AM	File folder	
Examples	7/1/2012 10:46 AM	File folder	
Help	7/1/2012 10:46 AM	File folder	
Icons	7/1/2012 10:46 AM	File folder	
include	7/1/2012 10:46 AM	File folder	
Lang	7/1/2012 10:46 AM	File folder	
lib	7/1/2012 10:47 AM	File folder	
libexec	7/1/2012 10:47 AM	File folder	
mingw32	7/1/2012 10:47 AM	File folder	
Packages	7/1/2012 10:47 AM	File folder	
Templates	7/1/2012 10:47 AM	File folder	
copying.txt	2/9/2002 9:32 AM	Text Document	19 KB
devcpp.exe	2/22/2005 8:22 PM	Application	942 KB
devcpp.map	2/22/2005 8:22 PM	Linker Address Map	1,744 KB
NEWS.txt	12/3/2004 9:15 PM	Text Document	11 KB
Packman.exe	2/20/2005 2:04 AM	Application	372 KB
Packman.map	2/20/2005 2:04 AM	Linker Address Map	682 KB
uninstall.exe	10/4/2010 9:13 PM	Application	72 KB

Step 2: When You Double Click .exe file the following window appears. Select English language and press 'Next'.



```
#include <w

/* Declare
LRESULT CAL

/* Make th
char szClas

int WINAPI
```

This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment options in the Tools menu.

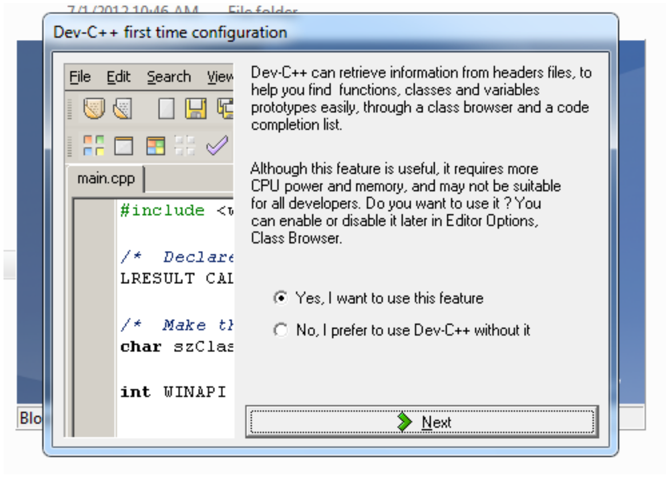
Select your language :
Chinese
Chinese (TW)
Croatian
Czech (Èeŀtina)
Danish
Dutch (Nederlands)
English (Original)

Select your Dev-C++ theme :
New Look ▾ Preview
☐ Use XP Theme

Next

Step 3:

A window appears asking the user about creating a cache.



```
#include <w

/* Declare
LRESULT CAL

/* Make th
char szClas

int WINAPI
```

Dev-C++ can retrieve information from headers files, to help you find functions, classes and variables prototypes easily, through a class browser and a code completion list.

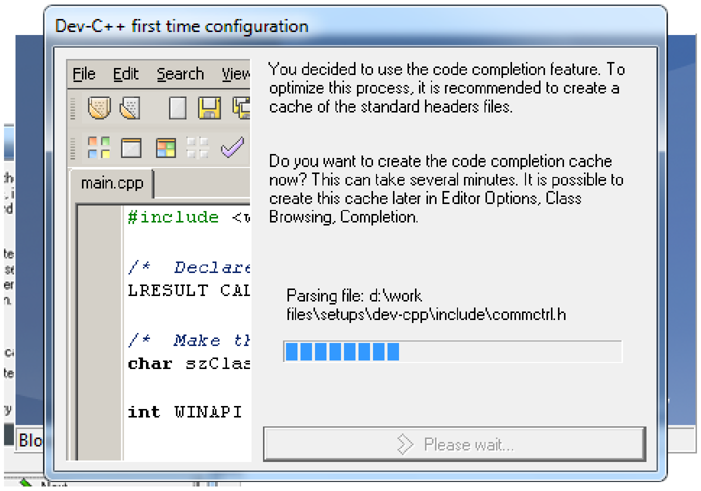
Although this feature is useful, it requires more CPU power and memory, and may not be suitable for all developers. Do you want to use it ? You can enable or disable it later in Editor Options, Class Browser.

[✓ Yes, I want to use this feature]
No, I prefer to use Dev-C++ without it

Next

Step 4:

Select ‘Yes’ and Press ‘Next’. The setup will proceed as follows



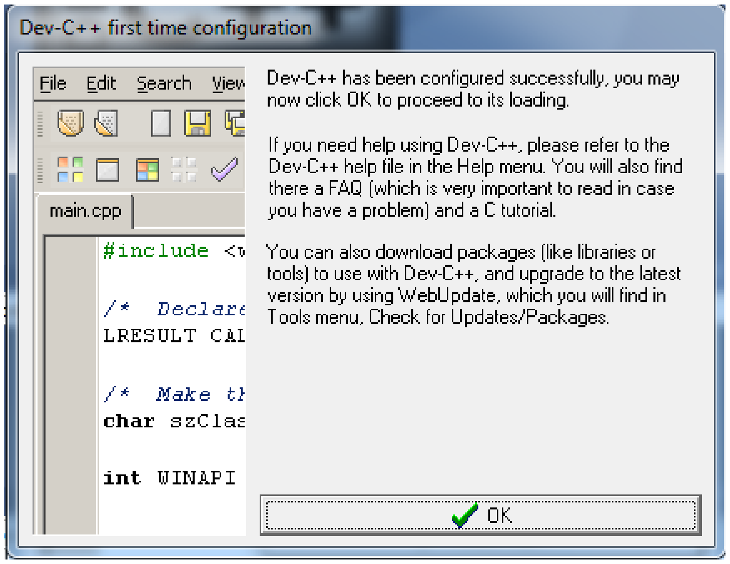
```
#include <w
/*  Declare
LRESULT CAL
/*  Make th
char szClas
int WINAPI
```

You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.

Do you want to create the code completion cache now? This can take several minutes. It is possible to create this cache later in Editor Options, Class Browsing, Completion.

Parsing file: d:\work files\setups\dev-cpp\include\commctrl.h

After the process is completed the following Window appears, that confirms the successful configuration of Dev C++. Click ‘OK



```
#include <w
/*  Declare
LRESULT CAL
/*  Make th
char szClas
int WINAPI
```

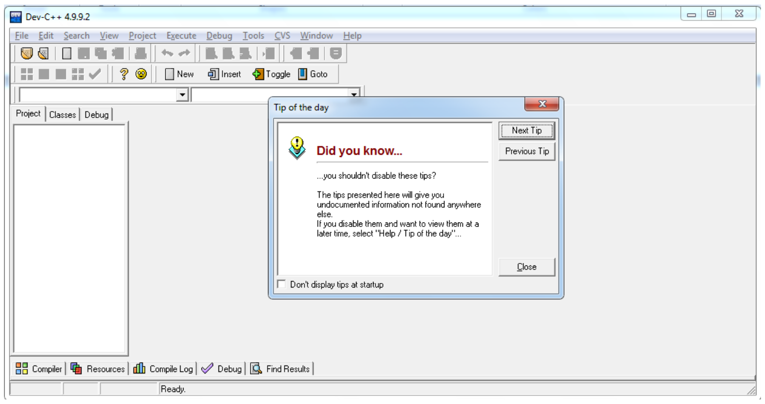
Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.

If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu. You will also find there a FAQ (which is very important to read in case you have a problem) and a C tutorial.

You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.

<u>Starting Dev C++:</u>

Starting with Dev C++ results in opening of following Window.


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Tip of the day
Next Tip
Previous Tip
Close
Compiler Resources Compile Log Debug Find Results
Ready.

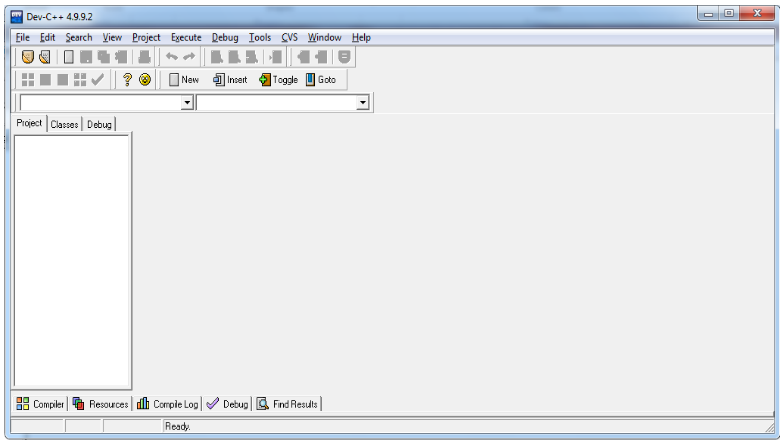
Did you know...

...you shouldn't disable these tips?

The tips presented here will give you undocumented information not found anywhere else.
If you disable them and want to view them at a later time, select "Help / Tip of the day"...

Don't display tips at startup

Click on 'Close' button. You will come up with IDE of Dev C++

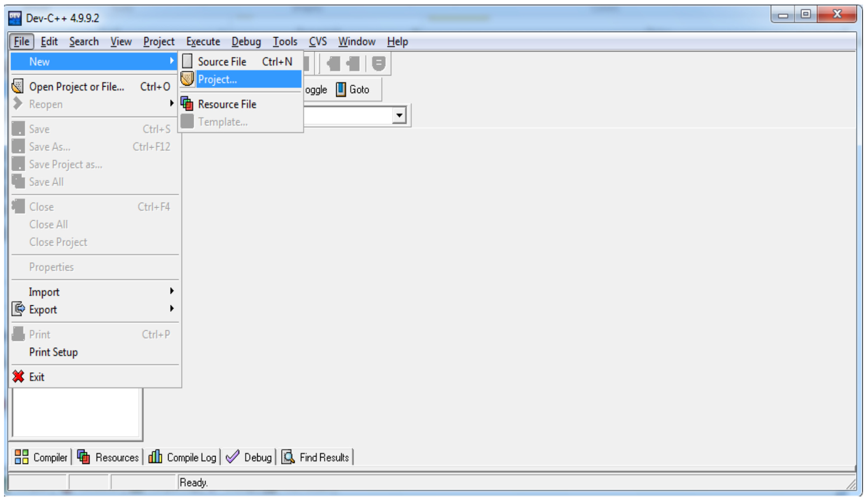

Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Compiler Resources Compile Log Debug Find Results
Ready.

<u>Project Creation:</u>

Step 1:

To create a new Project

File<New<Project



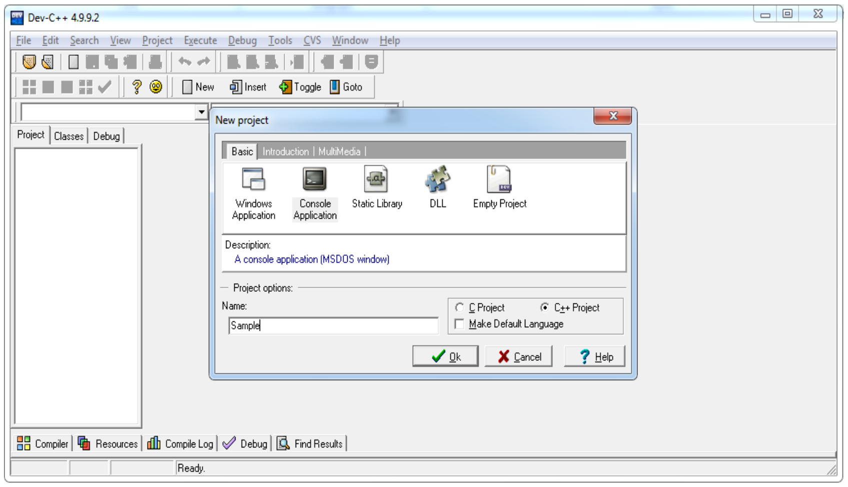
Step 2:

Following Window appears, asking about the Windows Application , Console Application …..

- Select Console Application

- Write the name of Project
- Select the ‘C++ Project’ radio button

Press OK



Description:
A console application (MSDOS window)

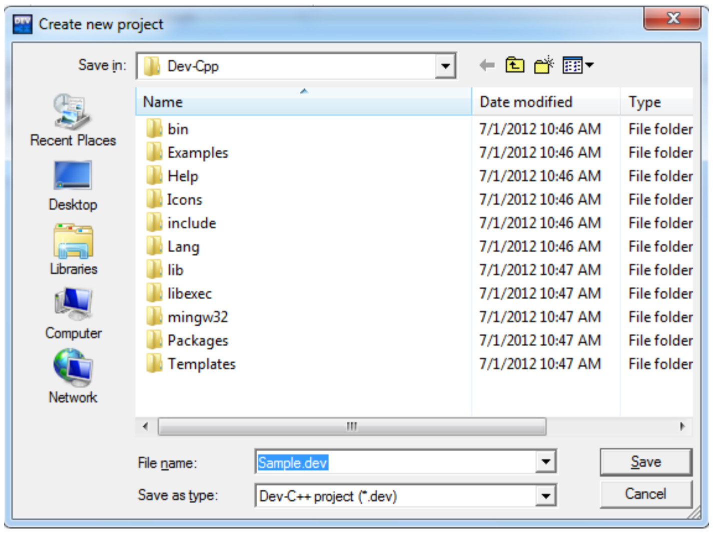
Project options:
Name: Sample
C Project [✓ C++ Project]
Make Default Language

Step 3:

Directory opens that enable the user to select the path of the project by browsing the directory. After Selecting the path select ‘Save’

Step 4:

The following ‘main.cpp’ file will open.



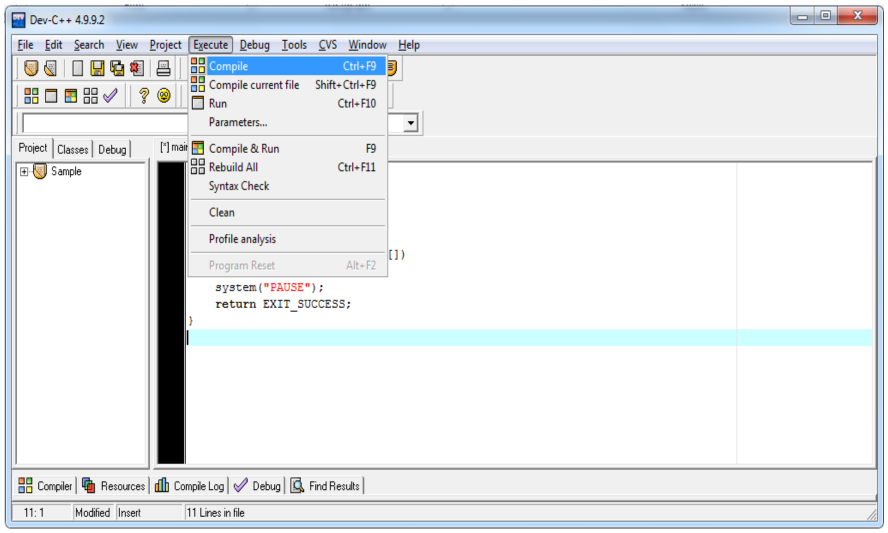
Save in: Dev-Cpp

Name	Date modified	Type
bin	7/1/2012 10:46 AM	File folder
Examples	7/1/2012 10:46 AM	File folder
Help	7/1/2012 10:46 AM	File folder
Icons	7/1/2012 10:46 AM	File folder
include	7/1/2012 10:46 AM	File folder
Lang	7/1/2012 10:46 AM	File folder
lib	7/1/2012 10:47 AM	File folder
libexec	7/1/2012 10:47 AM	File folder
mingw32	7/1/2012 10:47 AM	File folder
Packages	7/1/2012 10:47 AM	File folder
Templates	7/1/2012 10:47 AM	File folder

File name: Sample.dev
Save as type: Dev-C++ project (*.dev)

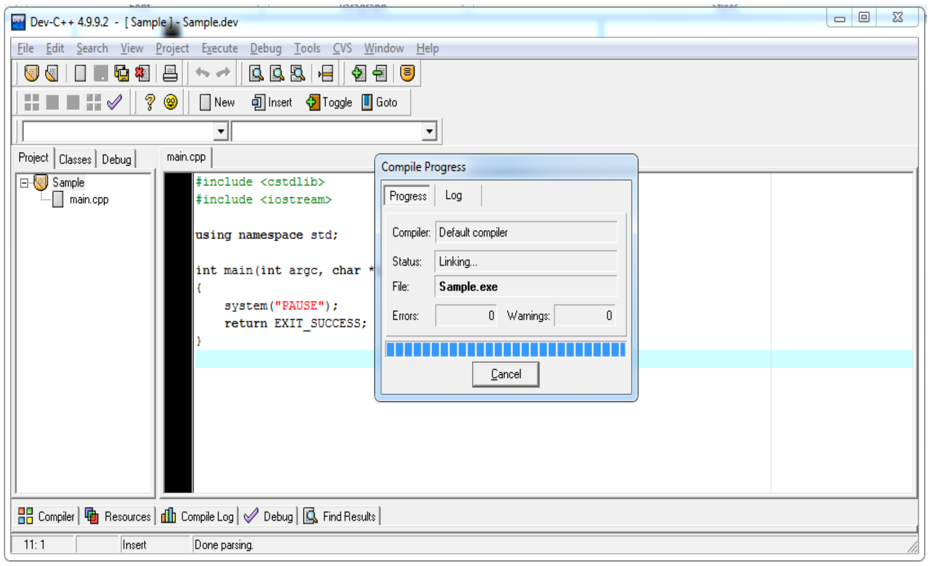
Compilation and Debugging of Project:

Step1: Now the project is compiled Execute<Compile


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
Compile Ctrl+F9
Compile current file Shift+Ctrl+F9
Run Ctrl+F10
Parameters...
Compile & Run F9
Rebuild All Ctrl+F11
Syntax Check
Clean
Profile analysis
Program Reset Alt+F2
Project Classes Debug
Sample
Compiler Resources Compile Log Debug Find Results
11: 1 Modified Insert 11 Lines in file

```
[])
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

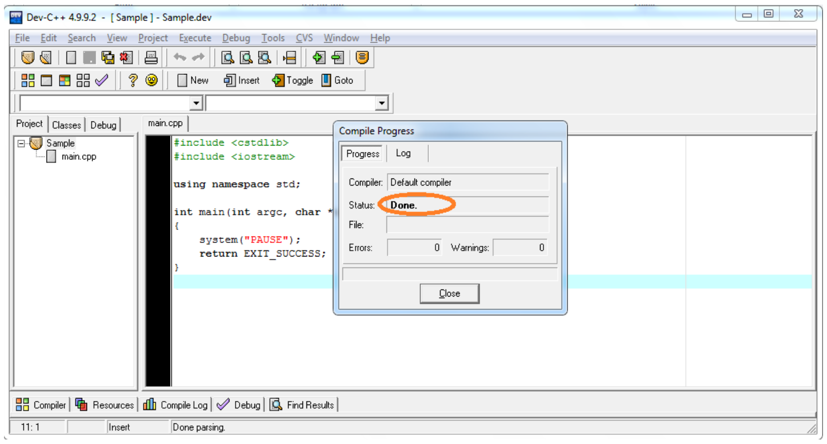
Following Window will open that enables the user to see number of ‘Errors’ and ‘Warnings’


Dev-C++ 4.9.9.2 - [Sample] - Sample.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
main.cpp
Sample
main.cpp
Compile Progress
Progress Log
Cancel
Compiler Resources Compile Log Debug Find Results
11: 1 Insert Done parsing.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *
{
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Compiler: Default compiler
Status: Linking...
File: Sample.exe
Errors: 0 Warnings: 0

After the Compilation is done Status ‘Linking’ is replaced by ‘Done’

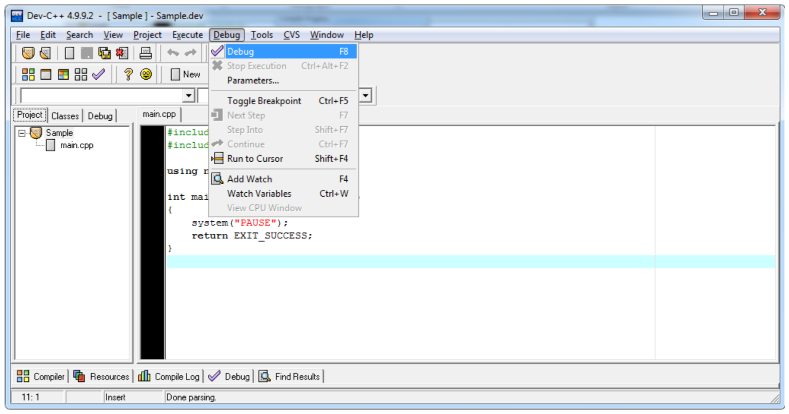

Dev-C++ 4.9.9.2 - [Sample] - Sample.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
main.cpp
Sample
main.cpp
Compile Progress
Progress Log
Close
Compiler Resources Compile Log Debug Find Results
11: 1 Insert Done parsing.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *
{
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Compiler: Default compiler
Status: Done.
File:
Errors: 0 Warnings: 0

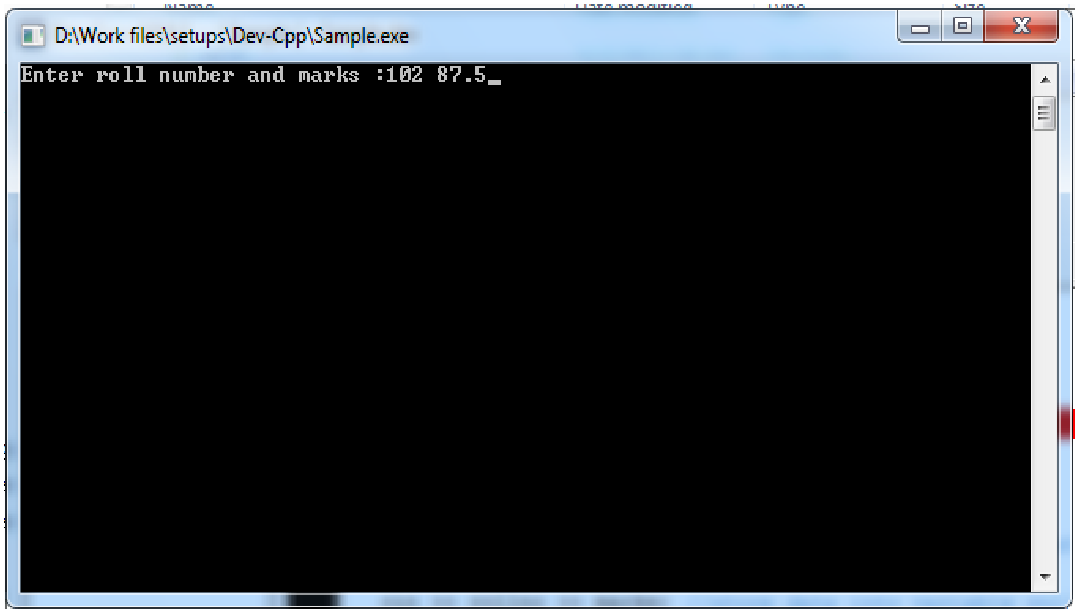
Step 2:

Now the project is Debugged


Dev-C++ 4.9.9.2 - [Sample] - Sample.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
Debug F8
Stop Execution Ctrl+Alt+F2
Parameters...
Toggle Breakpoint Ctrl+F5
Next Step F7
Step Into Shift+F7
Continue Ctrl+F7
Run to Cursor Shift+F4
Add Watch F4
Watch Variables Ctrl+W
View CPU Window
Project Classes Debug
main.cpp
Sample
main.cpp
Compiler Resources Compile Log Debug Find Results
11: 1 Insert Done parsing.

```
#includ
#includ

using n

int mai
{
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Debug<Debug Output will appear on ‘Console’



```
Enter roll number and marks :102 87.5_
```

Structure of C++ Program:

A **C++ program starts** with function called main ( ). The **body of the function** is **enclosed** between curly braces. The **program statements** are written within the braces. Each **statement must end** by a semicolon **;** (statement terminator).

A C++ program may contain as many functions as required. However, when the **program is loaded in the memory**, the control is handed over to function main ( ) and it is the first function to be executed.

```
#include<header file>
int main()
{
...........

}
```

```
// This is my first program is C++
/* this program will illustrate different components of
a simple program in C++ */

# include<iostream>

Using namespacestd;
intmain ( )
{

   ................

   ................

   return 0;  // My first program in C++
}
```

Various components of this program are discussed below:

Comments:

The line of the above program with '//' are comments and are ignored by the compiler. Comments are included in a program to make it more readable. If a comment is short and can be accommodated in a single line, then it is started with double slash sequence in the first line of the program. However, if there are multiple lines in a comment, it is enclosed between the two symbols /* and */.

#include <iostream>

The line in the above program that start with # symbol are called **directives** and are **instructions to the compiler**. The word include with '#' tells the compiler to include the file iostream into the file of the above program. File **iostream** is a header file needed for **input/ output** requirements of the program. Therefore, this file has been included at the top of the program.

int main ( )

The word **main** is a **function name**. The brackets ( ) with main tells that main ( ) is a function. The word int before main ( ) indicates that integer value is being returned by the function main

(). When program is loaded in the memory, the control is handed over to function main ( ) and it is the first function to be executed.

Curly bracket and body of the function main ( )

A C++ program starts with function called main (). The body of the function is enclosed between curly braces. The program statements are written within the brackets. Each statement must end by a semicolon, without which an error message in generated.

return 0;

This is a new type of statement, called a return statement. When a program finishes running, it sends a value to the operating system. This particular return statement returns the value of 0 to the operating system, which means “everything went okay!”

Input Output Commands in C++ :

C++ **supports** input/output statements which can be used to feed new data into the computer or obtain output on an output device such as: VDU, printer etc. The following C++ stream objects can be used for the input/output purpose.

1. **cin** console input
2. **cout** console output

cout is used in conjunction with << operator, known as insertion or put to operator. **cin** is used in conjunction with >>operator, known as extraction or get from operator.

cout<< “My first computer";Once this statement is carried out by the computer, the message "My first computer" will appear on the screen.

Examples:

```
cout <<"Output sentence";                    // prints Output sentence on screen

cout << 120;                                 // prints number 120 on screen

cout << x;                                   // prints the value of x on screen

int num;
cin>> num;
```

cin can be used to input a value entered by the user from the keyboard. However, the get from operator>> is also required to get the typed value from cin and store it in the memory location.

In the above segment, the user has defined a variable' num' of integer type in the first statement and in the second statement he is trying to read a value from the keyboard.

Sample Example:

```
/* this program illustrates how to
declare variable, read data and display data. */

#include <iostream>

usingnamespacestd;


int                                   main                                   (                                   )
{
  int rollno;//declare the variable rollno of type int

  float marks;//declare the variable marks of type float
  cout<<"Enter roll number and marks :";
  cin>>rollno>> marks; //store data into variable rollno & marks
  cout<<"Rollno: "<<rollno<<"\n";
  cout<<"Marks: "<< marks;
  return0;
}
```

Sample Run: In this sample run, the user input is shaded.

```
Enter roll number and marks: 102 87.5
Rollno: 102
Marks: 87.5
```

Lab Tasks:

1. Take a simple C++ Program and Compile it by removing the elements required in basic Syntax. And write your observations.
2. Look for more Header files in C++ (at least five) and write their functions
3. Write a program to print the following text:
   Hello in my university (empty line)
   I am a student in it faculty (empty line)
   Welcome.
4. Find and correct the errors in the following code.

```
#include <iostream>
 int main()
{
        cout<< "we are happy"
}
```

5. Find the errors in the following code.

```
#include <iostream>
using namespace std;
int main
{
        cout>> "c++ first exam/n";
        cout<<"\n END\t PROGRAM\t BYE;
}
```

6. Input 3 numbers from user and print their sum and product and average.

Lab Session 10: Basics of C++ (data types & escape sequences)

Objective: To understand about data types and escape sequences.

Data Types:

C++ supports a large number of data types. The built in or basic data types supported by C++ are integer, floating point and character. These are summarized in table along with description and memory requirement.

Type	Byte	Range	Description
int	2	-32768 to +32767	Small whole number
long int	4	-2147483648 to +2147483647	Large whole number
float	4	3.4x10-38 to 3.4x10+38	Small real number
double	8	1.7x10-308 to 1.7x10+308	Large real number
long double	10	3.4x10-4932 to 3.4x10+4932	Very Large real number
char	1	0 to 255	A Single Character

Variables:

It is a location in the computer memory which can store data and is given a symbolic name for easy reference. The variables can be used to hold different values at different times during the execution of a program.
To understand more clearly we should study the following statements:
Total = 20.00; in this statement a value 20.00 has been stored in a memory location Total.

Declaration of a variable
Before a variable is used in a program, we must declare it. This activity enables the compiler to make available the appropriate type of location in the memory.
float Total;

You can declare more than one variable of same type in a single statement
int x,y;

Initialization of variable
When we declare a variable its default value is undetermined. We can declare a variable with some initial value.
int a = 20;

Escape Sequences:

Escape sequences are character combinations starting with a backslash (\) and followed by a letter or digits inserted in a string literal to produce a modified output. For example, to produce a newline when outputting a string, we can use the \n escape sequence.

The following table shows a list of escape sequences and their corresponding effect:

Escape Sequence	Output
\a	**Bell (beep)**
\n	**Newline**
\r	**Carriage Return**
\t	**Horizontal tab**
\0	**Null Character**
\b	**Backspace**

Lab Tasks:

1. Practice all the Escape sequences in your program and observe the results
2. Write a simple C++ Program that Display your name, Father Name (in a new line) and end with a beep.
3. Display the message "This is a C program." with the words separated by tabs.

Expected output

```
This is my first program
This is a C program
This is a c
Program
This
Is
a
c
program
This	is	a	c	program
```

4. Write a program that will show the following output Using cout Statement

```
**************************************************
**         INTRODUCTION TO COMPUTING          **
**                Language C++                **
  **               DEV C++                **
**************************************************
```

5. Write a C++ program which takes two numbers input from the user and print their sum, difference, product, modulus and division results.

6. Write a program that will return the size of each data types .

Lab Session 11: Operators in C++

Objective: To understand about the concept of operators in C++ and their use.

Operators in C++:

Operators are special symbols used for specific purposes. C++ provides six types of operators.

1. Assignment operators
2. Arithmetical operators
3. Compound operators
4. Increment Decrement operator
5. Relational operators
6. Logical operators
7. Conditional operators
8. Comma operator

1. Assignment (=) operator

The assignment operator assigns a value to a variable.

```
a = 5;
```

This statement assigns the integer value 5 to the variable a.

For example, let us have a look at the following code - I have included the evolution of the content stored in the variables as comments:

```
// assignment operator

#include <iostream>
usingnamespace std;
int main ()
{
int a, b;       // a:?,  b:?
a = 10;         // a:10, b:?
b = 4;          // a:10, b:4
a = b;          // a:4,  b:4
b = 7;          // a:4,  b:7

cout<<"a:";
cout<< a;
cout<<" b:";
cout<< b;

return 0;
}
```

This code will give us as result that the value contained in a is 4 and the one contained in b is 7.

The following expression is also valid in C++:

```
a = b = c = 5;
```

It assigns 5 to the all three variables: a, b and c.

2. Arithmetic operator

The five arithmetical operations supported by the C++ language are:

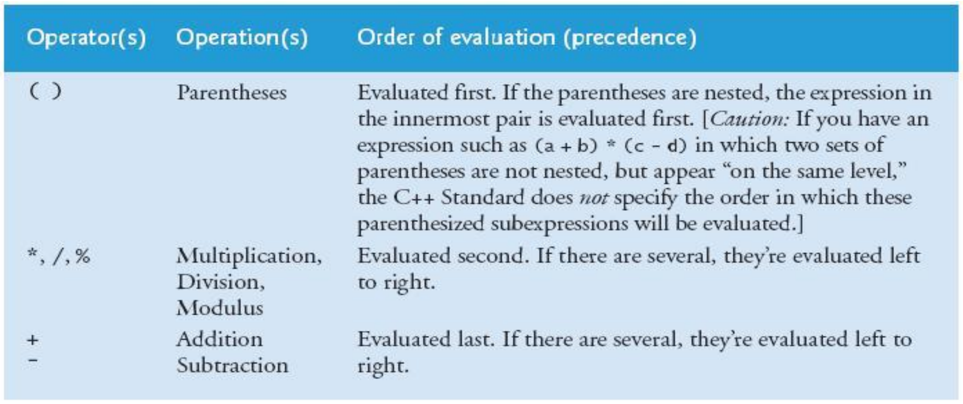
Operator	Meaning
+	Addition
_	Subtraction
*	Multiplication
/	Division
%	For remainder or modulus

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see is modulo; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

```
a = 11 % 3;
```

The variable ‘a’ will contain the value 2, since 2 is the remainder from dividing 11 between 3.

- The **precedence** of the arithmetic operation in C++ is as the following :



Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. [*Caution:* If you have an expression such as (a + b) * (c - d) in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does *not* specify the order in which these parenthesized subexpressions will be evaluated.]
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

3. Compound operators (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators:

For example:

```
// compound assignment operators
#include <iostream>
usingnamespacestd;
int main ()
{
int a, b=3;
  a = b;
  a+=2;          // equivalent to a=a+2
cout<< a;
return 0;
}
```

Expression	Is Equivalent To
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

4. Increase and Decrease (++, --)

Shortening even more some expressions, the increase operator (++) and the decrease operator (--). Increase or decrease by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:

```
c++;
c+=1;
c=c+1;
```

are all equivalent in its functionality: the three of them increase by one the value of c.

- The position of the ++ determines when the value is incremented.
- The position of the -- determines when the value is decremented.

Notice the difference:

Example 1	Example 2

B=3;
A=++B;
// A contains 4, B contains 4
A=--B;
// A contains 2, B contains 2
	B=3;
	A=B++;
	// A contains 3, B contains 4
	A=B--;
	// A contains 3, B contains 2

In Example 1, B is increased or decreased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased or decreased.

5. Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.
We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C++:

Operator	Meaning
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Here there are some examples:

```
(7 == 5)    // evaluates to false.
(5 > 4)     // evaluates to true.
(3 != 2)    // evaluates to true.
(6 >= 6)    // evaluates to true.
(5 < 5)     // evaluates to false.
```

Of course, instead of using only numeric constants, we can use any valid expression, including variables.

```
Suppose that a=2, b=3 and c=6,
(a == 5)    // evaluates to false since a is not equal to 5.
(a*b >= c)  // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c) // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a) // evaluates to true.
```

Be careful! The operator = (one equal sign) is not the same as the operator == (two equal signs), the first one is an assignment operator (assigns the value at its right to the variable at its left) and the other one (==) is the equality operator that compares whether both expressions in the two sides of it are equal to each other. Thus, in the last expression ((b=2) == a), we first assigned the value 2 to b and then we compared it to a, that also stores the value2, so the result of the operation is true.

6. Logical operators (!, &&, ||)

Examples:

```
!(5 == 5)   // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)   // evaluates to true because (6 <= 4) would be false.
!true// evaluates to false
!false// evaluates to true.
( (5 == 5) && (3 > 6) )  // evaluates to false ( true&& false ).
( (5 == 5) || (3 > 6) )  // evaluates to true ( true || false
```

Operator	**Meaning**
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true**.**
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest. Therefore, in this last example

((5==5)||(3>6)), C++ would evaluate first whether 5==5 is true, and if so, it would never check whether 3>6 is true or not. This is known as short-circuit evaluation, and works like this for these operators:

operator	short-circuit
&&	if the left-hand side expression is false, the combined result is false (right-hand side expression not evaluated).
||	if the left-hand side expression is true, the combined result is true (right-hand side expression not evaluated).

7. Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:
condition? result1: result2
if condition is true the expression will return result1, if it is not it will return result2.

```
7==5 ?4 : 3     // returns 3, since 7 is not equal to 5.
7==5+2 ?4 : 3   // returns 4, since 7 is equal to 5+2.
5>3 ?a : b      // returns the value of a, since 5 is greater than 3.
a>b ? a : b     // returns whichever is greater, a or b.
```

```
// conditional operator
#include <iostream>
usingnamespacestd;
int main ()
{
Int a,b,c;
  a=2;
  b=7;
  c = (a>b) ? a : b;
cout<< c;
return 0;
}
```

In this example a was 2 and b was 7, so the expression being evaluated (a>b) was not true, thus the first value specified after the question mark was discarded in favor of the second value (the one after the colon) which was b, with a value of 7.

8. Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example, the following code:

```
a = (b=3, b+2);
```

Would first assign the value 3 to b, and then assign b+2 to variable a. So, at the end, variable a would contain the value 5 while variable b would contain value 3.

<u>Lab Tasks:</u>

1. Show the value of x after each statement is performed. Write program to check your answers. Here x is an integer.
    a. x = 7 + 3 * 6 / 2 - 1;
    b. x = 2 % 2 + 2 * 2- 2 / 2;
    c. x = (3 * 9 * (3 + (9 * 3 / (3))));
    d. x = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8;
    e. x = 3 / 2 * 4 + 3 / 8 + 3;
2. Write a program to find the number of bytes occupied by various data types using the sizeof operator?
    a. int a;
    b. char b;
    c. float c;
    d. long int d;
    e. bool e;
    f. unsigned int j;
    g. unsigned long k;

    Hint: cout<< “size of integer variable” << sizeof(a)<< endl;

Lab Session 12: C++ Decision Making Statement

The if, if...else and nested if...else statement are used to make one-time decisions in C++ Programming, that is, to execute some code/s and ignore some code/s depending upon the test condition. Without decision making, the program runs in similar way every time. Decision making is an important feature of every programming language using C++ programming.

C++ if...else statement

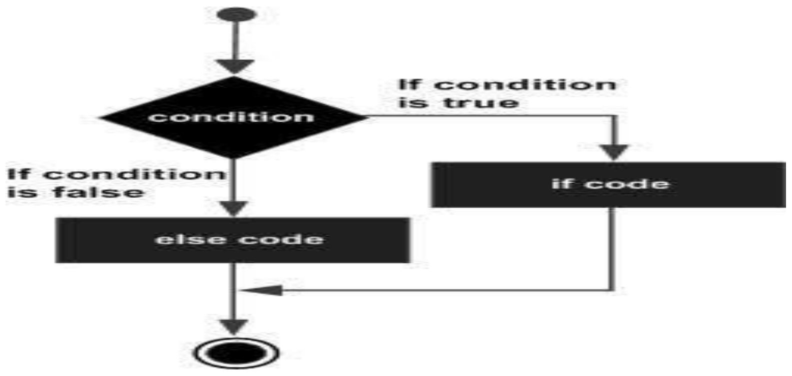
An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

Syntax:

The syntax of an if...else statement in C++ is:

```
if(boolean_expression)
{
   // statement(s) will execute if the boolean expression is true
}
else
{
  // statement(s) will execute if the boolean expression is false
}
```

If the boolean expression evaluates to **true**, then the **if block** of code will be executed, otherwise **else block** of code will be executed.



Flow Diagram

Example:

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

int main ()
{
   int a = 100;
   if( a < 20 )
   {
    cout << "a is less than 20;" << endl;
   }
   else
   {
       cout << "a is not less than 20;" << endl;
   }
   cout << "value of a is : " << a << endl;
   system("pause");
   return 0;
}
```

When the above code is compiled and executed, it produces the following

```
a is not less than 20;
value of a is : 100
```

The if...else if...else Statement:

An if statement can be followed by an optional else if...else statement, which is very usefull to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of he remaining else if's or else's will be tested.

Syntax:

```
if(boolean_expression 1)
{
   // Executes when the boolean expression 1 is true
}
else if( boolean_expression 2)
{
   // Executes when the boolean expression 2 is true
}
else if( boolean_expression 3)
{
   // Executes when the boolean expression 3 is true
}
else
{
   // executes when the none of the above condition is true.
}
```

Class Exercise:

1.Write a Program to check whether the integer entered by user is positive, negative or zero.

2.Write a program that will define the grade of student according to the numbers that the student gain in his examination where the grade system is define as following

a)A+ when the marks are 90 or above **b)A when the marks are 80 or above**

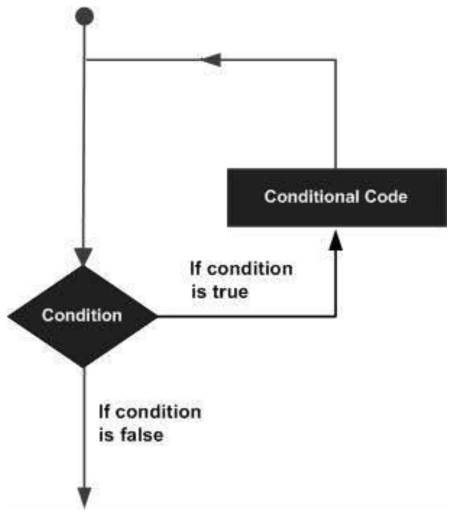
c)B when the marks are 70 or above **d)C when the marks are 60 or above**

e)D when the marks are 50 or above **f)E when the marks are 40 or above**

g)F otherwise

C++ Loop Types

There may be a situation, when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:



C++ programming language provides the following types of loop to handle looping requirements.

Loop Type	**Description**
While loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
For loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
DO....While loop	Like a while statement, except that it tests the condition at the end of the loop body
Nested loops	You can use one or more loop inside any another while, for or do..while loop.

While Loop

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

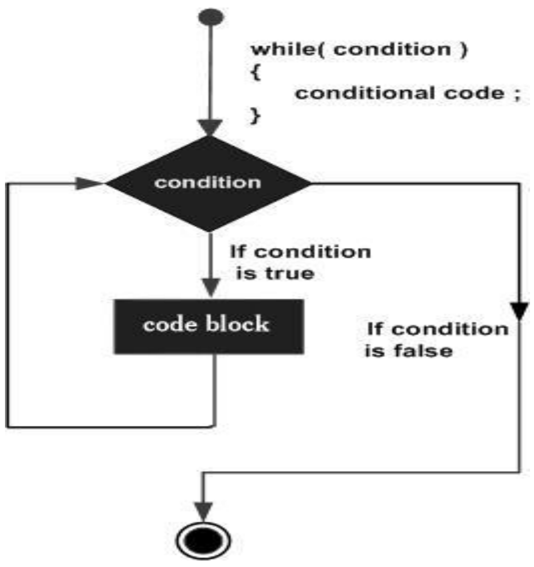
Syntax:

The syntax of a while loop in C++ is:

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

Here, **statement(s)** may be a single statement or a block of statements. The**condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram:



Here, key point of the *while* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
#include <iostream>
using namespace std;
int main ()
{
   int a = 10;
   while( a < 20 )
   {
       cout << "value of a: " << a << endl;
       a++;
   }
 System("pause");
   return 0;
}
```

Class Exercise:

3. Write a C++ program to find factorial of a positive integer entered by user. (Factorial of n = 1*2*3...*n)

4.Write a program that will execute a while loop number of times that is entered by user.

Lab Session 13:C++ Loops Types

There may be a situation, when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times.

C++ for loop

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

Syntax:

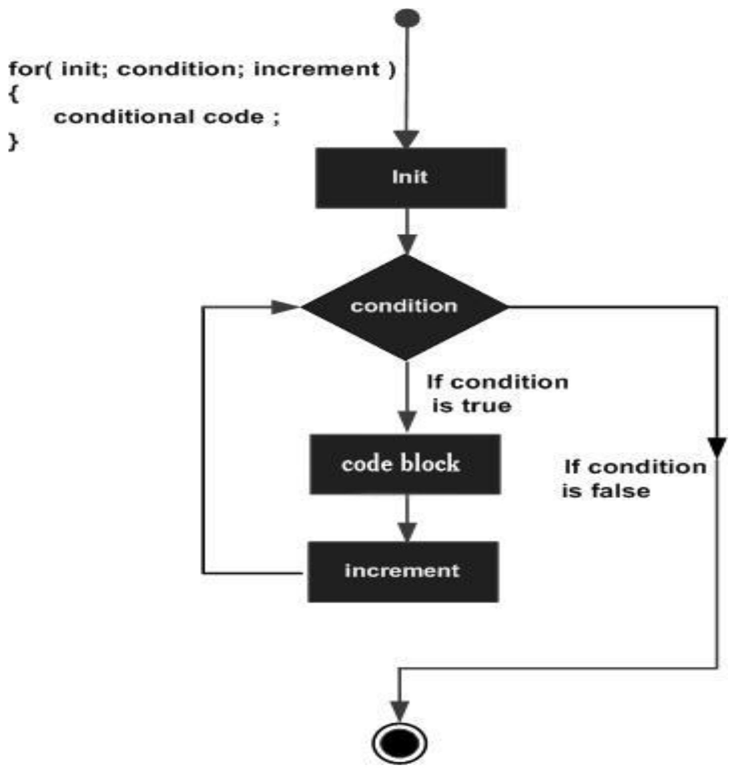
The syntax of a for loop in C++ is:

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

Here is the flow of control in a for loop:

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

Flow Diagram:



Example:

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

int main ()
{
  // for loop execution
  for( int a = 10; a < 20; a++)
  {
     cout << "value of a: " << a << endl;
  }

  return 0;
}
```

C++ 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, except that a do...while loop is guaranteed to execute at least one time.

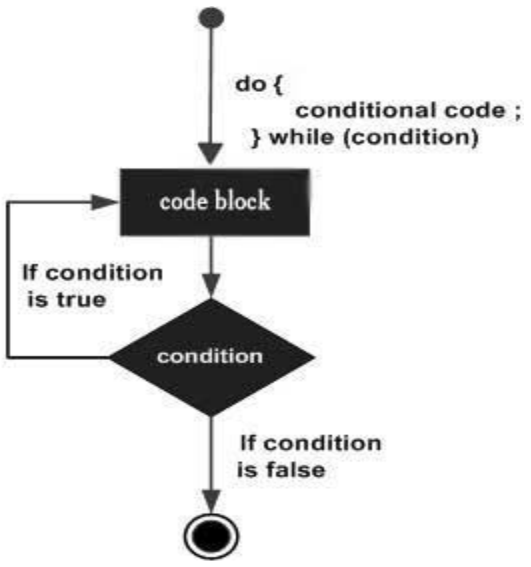
Syntax:

The syntax of a do...while loop in C++ is:

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

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

Flow Diagram:



Example:

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

int main ()
{
   int a = 10;
   do
   {
       cout << "value of a: " << a << endl;
       a = a + 1;
   }
while( a < 20 );
 System("pause");
   return 0;
}
```

CLASS EXERCISE:

1.Write a C++ program to add numbers entered by user until user enters 0.

2. Write a program that initialize the loop control variable with negative integer value and outputs the next ten values that are less than the loop control variable values.

3.Write a program that will print the table of user entered number using for loop..

4.Write a program that will display the user entered string and terminated the program when the string entered by the is equal to goodbye using do while loop ...

5.Write a program that count down from 10 to zero and the counter aborted when the value of counter become 3 display the message using for loop

10 ,9 ,8 ,7 ,6 ,5 ,4 ,3   counter aborted....

Lab Session: 14 Array In C++

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays:

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows:

```
typearrayName[arraySize];
```

This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement:

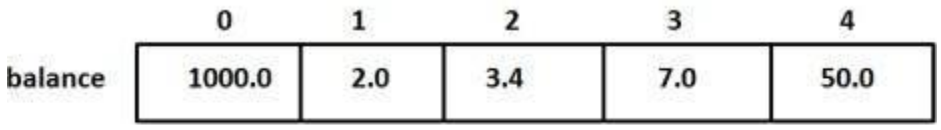
```
double balance[10];
```

Initializing Arrays:

```
double balance[]={1000.0,2.0,3.4,17.0,50.0};
```

```
balance[4]=50.0;
```

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representation of the same array we discussed above:



Accessing Array Elements:

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

```
double salary = balance[9];
```

The above statement will take 10th element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays:

Example:1

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

int main()
{
		double distance[] = {44.14, 720.52, 96.08, 468.78, 6.28};

		cout<< "2nd member = " <<distance[1]<< endl;
		cout<< "5th member = " <<distance[4]<< endl;

		return 0;
}
```

Example:2

```
#include<iostream>
usingnamespacestd;
#include<iomanip>
usingstd::setw;
int main ()
{
int n[10];// n is an array of 10 integers
// initialize elements of array n to 0
for(inti=0;i<10;i++)
{
n[i]=i+100;// set element at location i to i + 100
}
cout<<"Element"<< setw(13)<<"Value"<< endl;


// output each array element's value
for(int j =0; j <10; j++)
{
cout<< setw(7)<< j << setw(13)<< n[ j ]<< endl;
}


return0;
}
```

C++ Multi-dimensional Arrays

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration:

```
typena    me[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional 5 .10 . 4 integer array:

```
intthreedim[5][10][4];
```

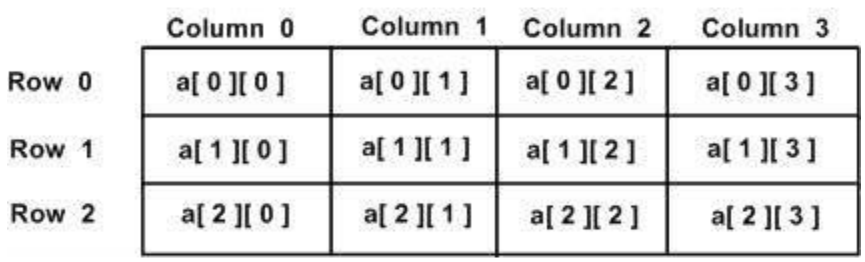
Two-Dimensional Arrays:

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows:

```
typearrayName[ x ][ y ];
```

Where type can be any valid C++ data type and arrayName will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array **a**, which contains three rows and four columns can be shown as below:



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form **a[i][j]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initializing Two-Dimensional Arrays:

Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4]={
{0,1,2,3},/*  initializers for row indexed by 0 */
{4,5,6,7},/*  initializers for row indexed by 1 */
{8,9,10,11}/*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example:

```
int a[3][4]={0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements:

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

```
intval= a[2][3];
```

The above statement will take 4th element from the 3rd row of the array. You can verify it in the above digram.

Example:

```
#include<iostream>

usingnamespacestd;


int main ()

{
```

```
int a[5][2]={{0,0},{1,2},{2,4},{3,6},{4,8}};

for(inti=0;i<5;i++)

for(int j =0; j <2; j++)

{

cout<<"a["<<i<<"]["<< j <<"]: ";

cout<< a[i][j]<< endl;

}


return0;

}
```

When the above code is compiled and executed, it produces the following result:

```
a[0][0]:0

a[0][1]:0

a[1][0]:1

a[1][1]:2

a[2][0]:2

a[2][1]:4

a[3][0]:3

a[3][1]:6

a[4][0]:4

a[4][1]:8
```

Class Exercise:

1. Write a program that will initialize an array of 10 elements and accessed value of each elements.

2. Write a program that the two arrays fills the elements of the arrays by user and sum these array .

3. Write a program that sums the all elements of the array.

4.Write a program that will initialize the 2-D array of [3][3] and accessed each element of 2-D array.