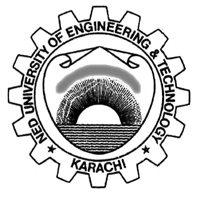
**PRACTICAL WORK BOOK**

**Computer and Programming (EE-163)**

**For**

**FE (ELECTRICAL)**

|  |
| --- |
| Name: |
| Roll Number: |
| Class: Semester: |
| Batch: |
| Department : |



**Department of Electrical Engineering  
NED University of Engineering & Technology, Karachi**

**CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab.**  **No.** | **Title of Experiments** | **Dated** | **Remarks** |
| 1 | Getting Started – Familiarization with Environment |  |  |
| 2 | C++ Building Blocks |  |  |
| 3 | C++ Mathematics |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Lab Session 01** |

# **Objective:**

# **Getting Started – Familiarization with Environment.**

In this lab session, we shall cover the following objectives

* How to install Code::Blocks IDE on computer
* Use Code::Blocks IDE and built in MinGW GCC Compiler to run our first program
* Explore Command Prompt (cmd)
* Run our first program via cmd
* Run an existing program (GuessNumber.exe) via cmd
  1. **Installing Code::Blocks Integrated Development Environment (IDE)**

C++ (pronounced cee plus plus) is a compiled language. In order to get started, two requirements are essential. First is the compiler and second is text editor (for typing the program). These requirements often come under a single packaged software application termed as Integrated Development Environment (IDE). For the lab sessions of this course we shall be using an open source and free of cost IDE called **Code::Blocks**. Getting Code::Blocks is just a matter of few clicks (provided you have an internet connection). In order to download the IDE follow these steps

1. Access [www.codeblocks.org/downloads](http://www.codeblocks.org/downloads) from your favorite web browser.
2. Click ***Download the binary release***
3. Download the Code::Blocks with Mingw setup file, at the time of writing this text codeblocks-16.01mingw-setup.exe was available.
4. You are ready to go now.

Note: These instructions are for Windows users. If you are running any other operating system then download the version for your operating system.

*If you don’t have internet access you can get a copy of the binary release from the Computer Lab. For now, it’s only available for Windows users.*

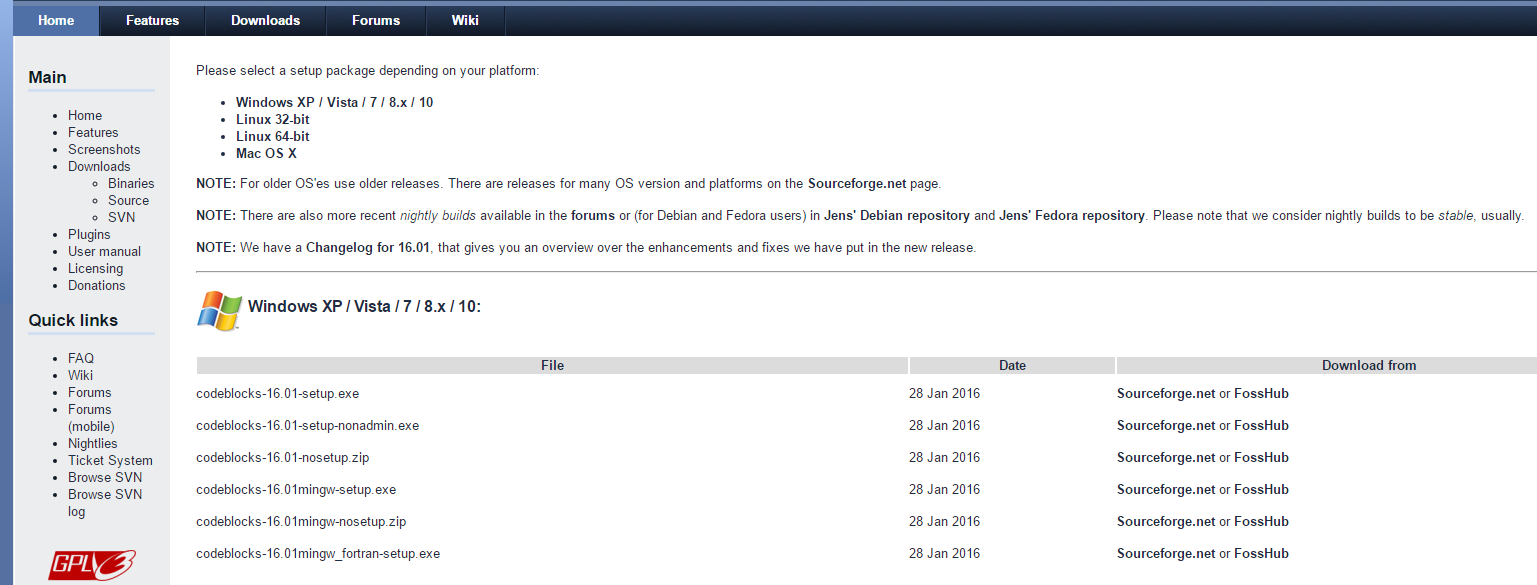


Figure 1 Screenshot of Downloads Page for Binary Release

Installation process is simple. Run the executable file you just downloaded (or acquired from Computer Lab). The installation Wizard will guide you through the whole process.

Once you run the setup file, the Wizard will get started.

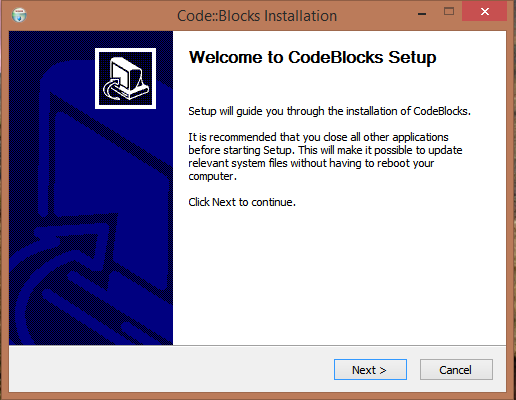


Figure 2 Step 1 of Installation Process Wizard Guide

Click Next to continue.

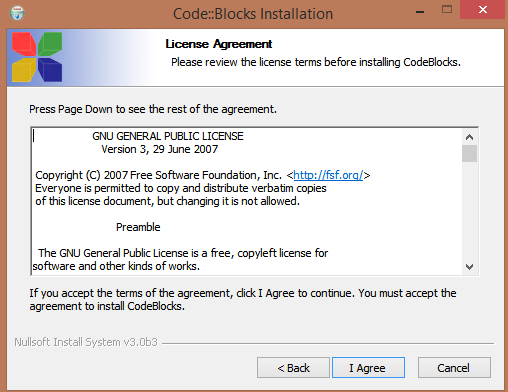


Figure 3 Step 2 of installation process License Agreement

You must agree with license terms to install and use Code::Blocks (read the terms provided and click I Agree). Once you are agreed with the terms, the installation wizard will now prompt to choose the components to install, check all components and click Next.

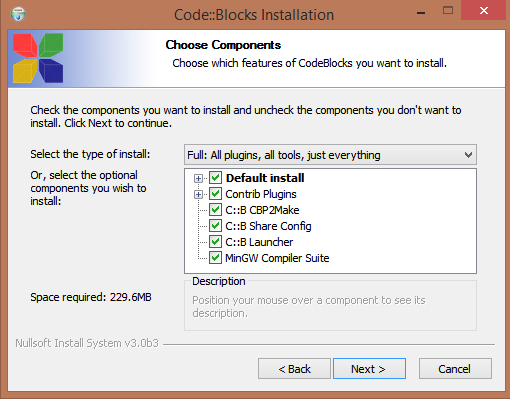


Figure 4 Step 3 of installation process, Components to install

Now select the hard disk location to install the Code::Blocks (using default is recommended)

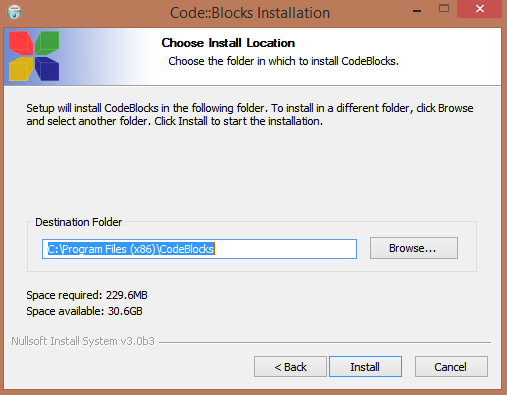
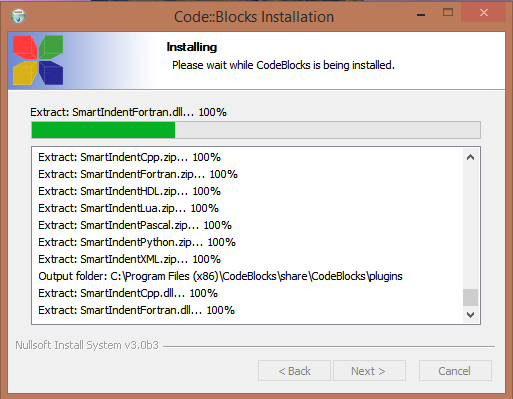


Figure 5 Step 4 of installation process, choose destination

Once you click the install button the installation will take place. Upon successful installation you will get

the message.

Figure 6 Installation in process

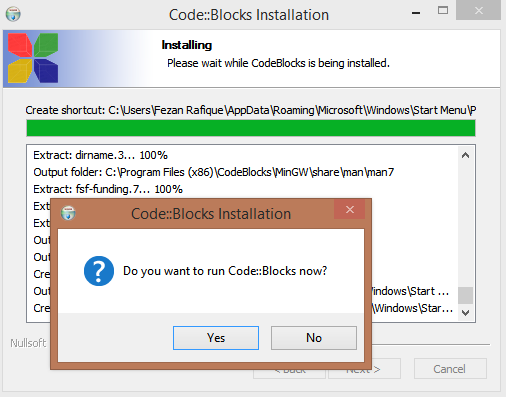


Figure 7 Installation successful

Once the installation process completed, click Finish button

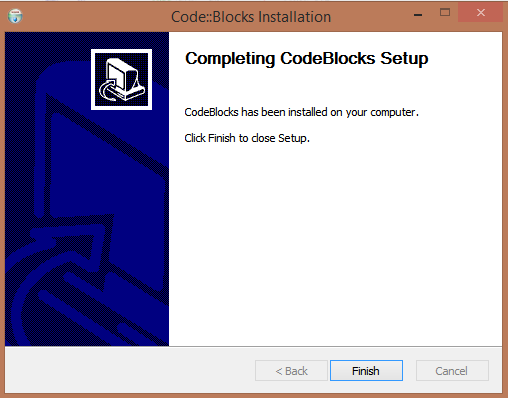


Figure 8 Installation process completed

* 1. **Running the First Program**

Once, the IDE is installed successfully we are now ready to develop our first C++ program. Follow the following steps

* Open Code::Blocks
* Create a new empty file (shortcut Ctrl + Shift + N )
* Save the file as lab\_01\_code\_01.cpp
* Beware about the format .cpp

**lab01\_code\_01.cpp**

#include<iostream>

using namespace std;

int main(void)

{

cout<<"Hello World";

return 0;

}

* Type the code as shown (don’t worry if you don’t understand it for now)
* After typing the code Go to BUILD>>BUILD and RUN (shortcut F9)
* If your program was successfully written, it will be executed otherwise you will get an error

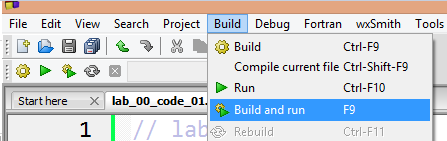


Figure 9 Step to Build Code

Screen Clipping

Figure 10 Console Log for Successful Build

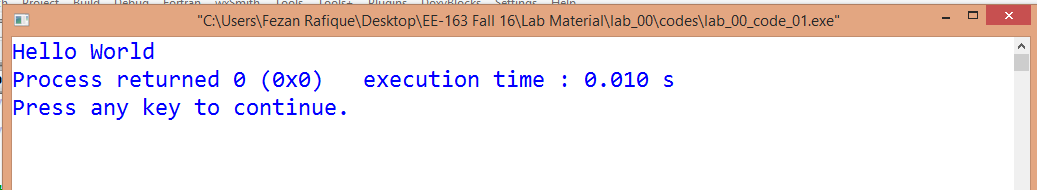


Figure 11 Output for lab\_01\_code\_01.cpp

* 1. **Exploring Command Prompt**

The target of our first program and all the other programs in this course is Console (command prompt or terminal). It is therefore necessary to have a brief introduction of command prompt.

To start command prompt, type “**cmd**” (without quotes) in Run.

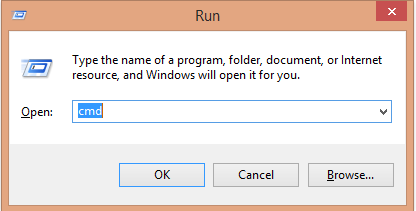


Figure 12 Run command for command prompt

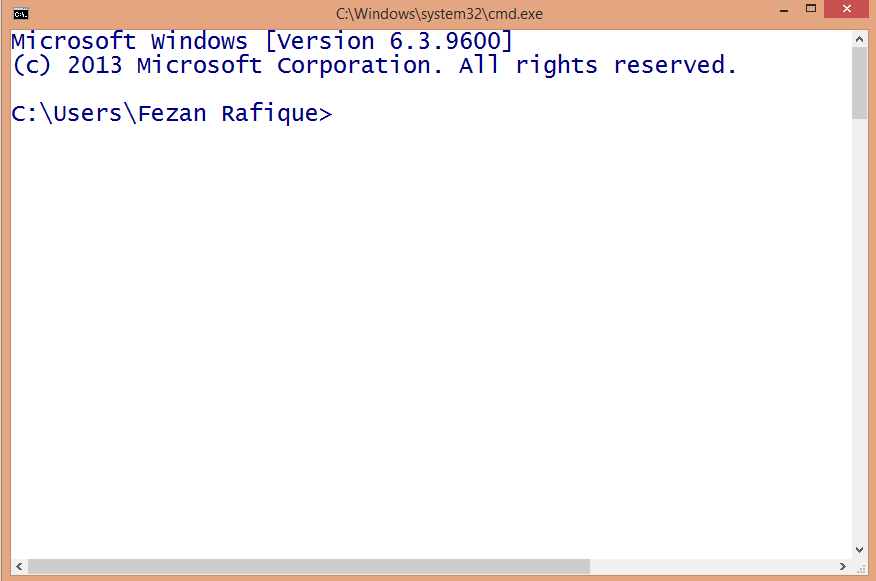
This will open the command prompt window. 

Figure 13 Command Prompt

To navigate through the directories, one can use **cd** command. A sample is shown in figure.

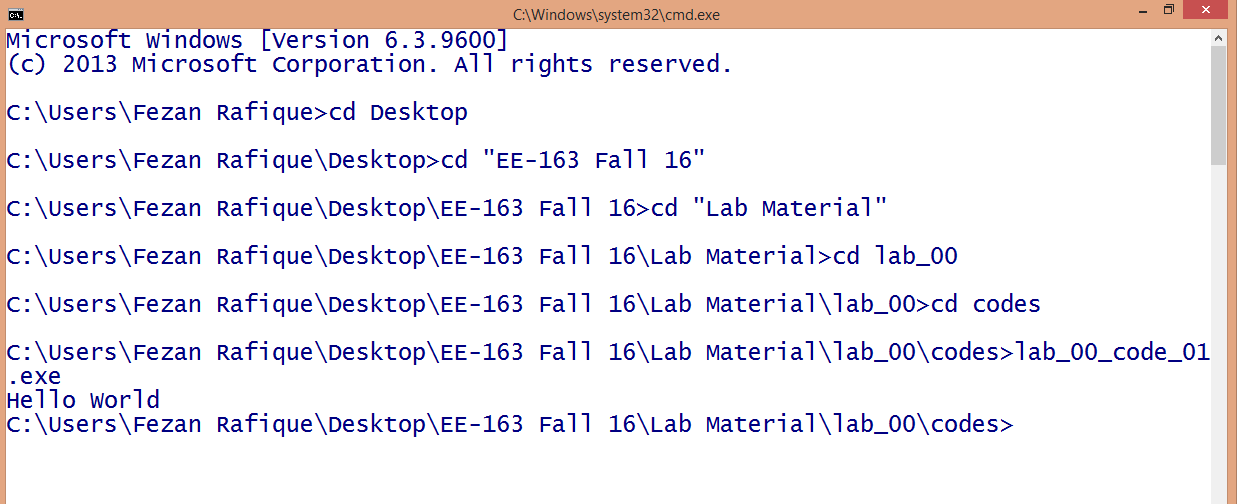


Figure 14 Navigating directories

There are many useful commands for command prompt, following links are helpful to get started.

* http://www.digitalcitizen.life/command-prompt-how-use-basic-commands
* http://www.computerhope.com/overview.htm
  1. **Run GuessNumber.exe**

As part of cmd exercise we shall now run an already developed program called **GuessNumber.exe** through cmd. This file is provided in the folder for Lab01

* GuessNumber.exe is already written program, the program asks the user to guess a number (which is in computer’s mind)
* The user will respond by typing and can do so, until correct number is guessed
* In the meanwhile for any wrong guess computer will give a hint
* Let’s try it

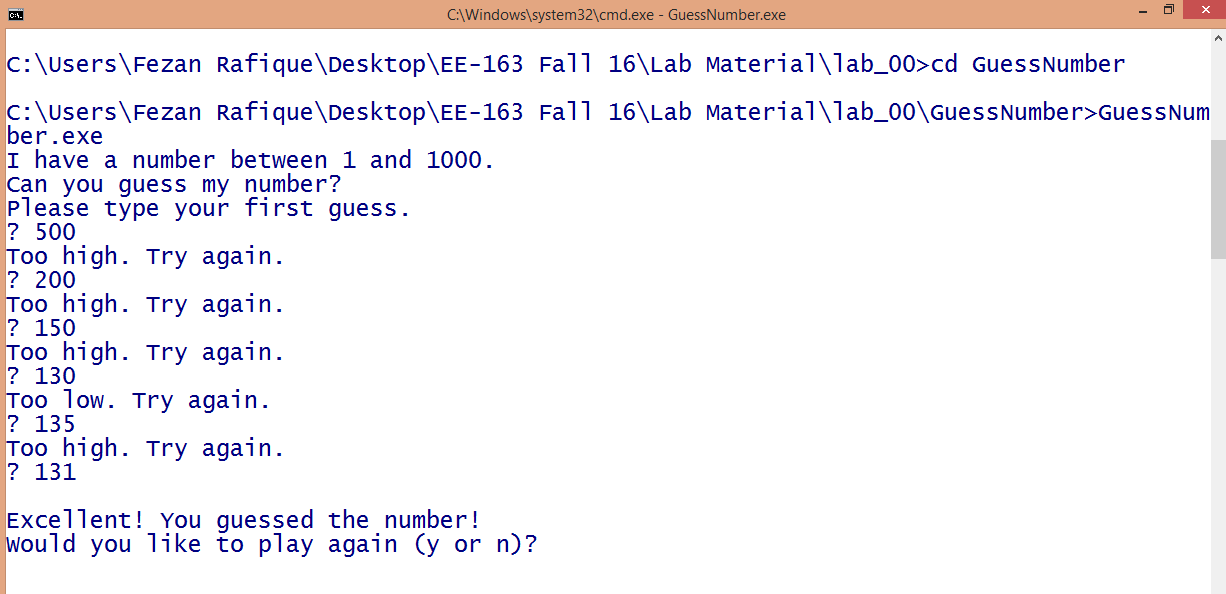


Figure 15 Running GuessNumber.exe

# Exercise

**Task 1:**

Write a program to print text in following pattern,

Hello World

Hello World

Hello World

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| **Lab Session 2** |

# **Objective**:

# C++ Building Blocks

In this lab session, we shall cover the following objectives

* Basic data types in C++
* Declaring and using variables
* Comments in a C++ Program
* Printing variable values with cout
* Interactive computing with cin
* Escape sequences

# **2.1 Basic Data Types in C++**

# Fundamental to any computer program is the data associated with its use. Based on the nature of data it can be classified into various categories. Data types are important to understand, they define proper use of an identifier and expression. In C++ data types can be categorized as following.

Figure Basic data types in C++

*Numeric*: This type contains the numbers including integers and floating point values. Following are the example of numeric data

* 100
* 895
* -237
* 6.022140857 × 10 ^ 23
* 6.62607004 × 10 -34
* -1.60217662 × 10-19

*Character*: Character data includes the alpha numeric characters and special symbols (enclosed in single quotes). Following are the examples

* ‘a’
* ‘F’
* ‘@’
* ‘%’
* ‘^’

*Strings*: Strings include all the text values (enclosed in double quotes). Following are the examples

* “Finland”
* “NED University”
* “PO Box No 341”
* “all along the watch tower”

*Boolean*: Boolean includes true and false values.

Following C++ statements show the possible use of these data types

cout<<100;

cout<<‘~’;

cout<<true;

cout<<“Mixing the stream ”<<200<<‘#’<<true<<“ ”<<false;

**2.2 Declaring and Using Variables**

* Variables are named objects with a specific type
* Variables can be used to store data of a certain type which can later be used, processed and/or updated in the program
* A variable must be declared using appropriate keyword
* There are some rules with variable naming

The following table shows the keyword and memory requirement of several data types

|  |  |  |
| --- | --- | --- |
| **Type** | **Keyword** | **Memory** |
| **Boolean** | bool | 1 Byte |
| **Character** | char | 1 Byte |
| **Integer** | int | 4 Bytes |
| **Floating point** | float | 4 Bytes |
| **Double floating point** | double | 8 Bytes |
| **String** | string | ? |

lab\_02\_code\_01.cpp

Following code can be used to check the memory requirements of various data types

#include<iostream>

using namespace std;

// sizeof() function calculates the Bytes

int main(void)

{

cout<<"Integer Bytes="<<sizeof(int);

cout<<"\nDouble Bytes="<<sizeof(double);

cout<<"\nCharacter Bytes="<<sizeof(char);

cout<<"\nBoolean Bytes="<<sizeof(bool);

return 0;

}

Identify the data types for the following items

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Type | Item | Type |
| TRUE |  | @ |  |
| 127 |  | 192.12 |  |
| Pakistan |  |  |  |

Variable Naming Rules: Following rules must be taken care while assigning a name to any variable.

* Variable name must start with a letter or \_ (underscore)
* May contains letter, numbers and the underscore character only
* Uppercase and lower case are distinct
* Name should not be a reserved keyword

Good Examples

salary, new\_name, myValue

Bad Examples

3name, my name, my-val, class, struct, while

A variable can be assigned a value with the assignment operator “=” . (Discussion about associativity will be the part of Lab03)

The following codes will be helpful to understand the use and role of variables in a C++ program

lab02\_code\_02.cpp

#include<iostream>

using namespace std;

int main(void)

{

int age; // declaring int variable

string name;// declaring string variable

float height\_in\_cms, weight\_in\_kg; // 2 float variables

age = 19; // now assigning values to variables

name = "Ahmed Khan";

height\_in\_cms = 123.8;

weight\_in\_kg = 58.7;

cout<<"Name:"<<name<<"\t Age:"<<age<<endl;

cout<<"Height(cm):"<<height\_in\_cms<<"\t Weight(kg):"<<weight\_in\_kg;

return 0;

}

lab02\_code\_03.cpp

#include<iostream>

using namespace std;

int main(void)

{

int Roll\_No = 123, salary = 40000;

float CGPA = 3.2;

double pi = 3.1214, x = 0.012, y;

string enrolment\_no = "ned/0145/14-15",name;

char section = 'D';

bool logical = 1;

cout<< "My Roll No:" <<Roll\_No<<"\t Pi="<<CGPA;

cout<< endl<<"Value of y is:"<<y<<endl;

cout<< "Name:"<<name<< endl;

cout << "Enrolment:"<< enrolment\_no;

return 0; }

**2.3 Comments in a C++ Program**

Program comments are explanatory statements that you can include in the C++ code that you write and helps anyone reading it's source code. All programming languages allow for some form of comments. C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.

C++ comments start with /\* and end with \*/. For example:

/\* This is a comment \*/

/\* C++ comments can also

\* span multiple lines

\*/

A comment can also start with //, extending to the end of the line. For example:

#include <iostream>

using namespace std;

main() {

cout << "Hello World"; // prints Hello World

return 0;

}

When the above code is compiled, it will ignore // prints Hello World and final executable will produce the following result:

Screen Clipping

Within a /\* and \*/ comment, // characters have no special meaning. Within a // comment, /\* and \*/ have no special meaning. Thus, you can "nest" one kind of comment within the other kind. For example:

/\* Comment out printing of Hello World:

cout << "Hello World"; // prints Hello World

\*/

**2.4 Idea of Interactive Computing**

In the above programs the value was directly assigned to the variable via assignment operator. This was done by the programmer. If it is needed to take input from the user and assign the user value to a particular variable. This is called interactive computing. C++ provides means to do so. One can use stream insertion via cin to assign value to a variable. This can be done like following

int value;

cout<<“Please enter the value ”;

cin>>value;

The following code further illustrates the idea of interactive computing

lab\_02\_code\_04.cpp

#include<iostream>

using namespace std;

int main(void)

{ // Starting braces of main

//\*\*\*Variable Declaration\*\*\*

string name, year, department ;

char section;

int roll\_no;

float cgpa;

//\*\*\*\*Taking user input\*\*\*\*

cout<<"Enter your name:";

cin>>name;

cout<<"Enter your Roll No.:";

cin>>roll\_no;

cout<<"Enter your department:";

cin>>department;

cout<<"Enter year of study:";

cin>>year;

cout<<"Enter your section:";

cin>>section;

cout<<"What is your CGPA?";

cin>>cgpa;

cout<<endl<<endl;

//\*\*\*\*\*Printing Output\*\*\*\*\*

cout<<"\t My Profile"<<endl;

cout<<"Name:"<<name<<"\tRoll No:"

<<roll\_no<<endl<<"Section:"

<<section<<"\tYear:"<<year<<

endl<<"Department:"<<

department<<"\tCGPA:"<<cgpa;

return 0;

}

**2.4 Escape Sequences**

You must have observed some difference in the last code, e.g. using \t in cout statements. This is called escape sequence. Escape sequences are used to represent certain special characters within string literals and character literals. Following escape sequences are commonly used in C++.

|  |  |
| --- | --- |
| **Sequence** | **Purpose** |
| \n | Next line |
| \r | Carriage return |
| \t | Horizontal tab |
| \b | Backspace |
| \a | Alert (beep) |
| \\ | Print \ |
| \’ | Print ’ |
| \” | Print “ |

Taking help from your textbook and online resources, try to figure out the purpose of these escape sequences and explain with the help of an example program.

# Exercise

**Task 1:**

How to insert single line and multiline comments in a C++ program.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task 2:**

Variable Declarations can appear almost anywhere in the body of C++ function (T/F).

If true, then discuss the situation in which variable declaration must be done prior to some specific task. Support you answer by giving example.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task 3:**

Calculate the maximum and minimum number that can be accommodated by *int* data type (calculate range).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task 4:**

What do you mean by *Variable Declaration* and *Variable Definition* in C/C++?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task 5:**

Check the output of the following *cout* functions and write your comments.

1. cout << “I am a computer geek, \rits a \blie.”
2. cout <<"a"<<"\t"<<"b"<<"\t"<<"c"<<endl;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Task 6:**

Temperature can be converted from Centigrade to Fahrenheit using following formula.

**F = (9.0/5.0)\*C + 32.0**

Write a program that ask user to input the temperature in degree Centigrade and calculates and displays the equivalent Fahrenheit value. Try the program with two possible data types, integer and float. Discuss the difference in result for the two programs.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| Lab Session 3 |

# **Objective:**

# **C++ Mathematics**

In this lab session, we shall cover the following objectives

* Mathematical Operators in C++
* Operators Precedence and Associativity
* Special Mathematics Operators
  + Increment/ Decrement
  + Compound Assignments
* Type Conversion
* <cmath> Library

**3.1 Mathematical Operators in C++**

C++ can be used to perform basic mathematical operations. The following program can be used to illustrate this.

Code 01

1. #include<iostream>
2. using namespace std;
3. int main()
4. {
5. int number1;
6. int number2;
7. int result;
8. cout<<"Please enter number1 & number2";
9. cin>>number1>>number2;
10. result = number1 + number2; // addition
11. result = number1 - number2; // subtraction
12. result = number1 \* number2; // multiplication
13. result = number1 / number2; // division
14. result = number1 % number2; // remainder division
15. cout<<result;
16. return 0;
17. }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **number1** | **number2** | **operation** | **result** |
| 1 | 12 | 8 | + | 20 |
| 2 | 12 | 8 | - | 4 |
| 3 | 12 | 8 | \* | 96 |
| 4 | 12 | 8 | / | 1 |
| 5 | 12 | 8 | % | 4 |

In the above program the variable number1 and number2 are called operands and they are connected via different operators in expressions given on line numbers 10 through 14. Response of each operation is stored in the variable result.

Keep in mind that modulus (%) operator is only defined for the data type integers

It is important to emphasis that result of the division is not as we expect in general. This is because the data type of number1 and number2 is integer, an integer divided by an integer will give an integer response, while truncating the decimal part of the value. This makes the order of precedence of arithmetic operators very significant.

The following code will help you develop intuition of C++ Mathematics

**Code 02**

#include<iostream>

using namespace std;

int main(void)

{ // BMI Calculator

float weight\_in\_kg ,height\_in\_meter ,bmi;

cout<<"\t \t \*\*Body mass index (BMI) calculator\*\* \n";

cout<<"\t Calculates an index that indicates"<<

" healthy weight distribution\n";

cout<<"Enter your weight in Kgs: ";

cin>> weight\_in\_kg;

cout <<"\nEnter your height in meters: ";

cin>> height\_in\_meter;

bmi=weight\_in\_kg/(height\_in\_meter\*height\_in\_meter);

cout<<"\nYour BMI value is:"<< bmi;

cout<<"\n\n \t\t Standard BMI Values for comparison \n";

cout<<"\n \t\t Less than 18.5 : Underweight";

cout<<"\n \t\t Between 18.5 and 24.9 : Normal";

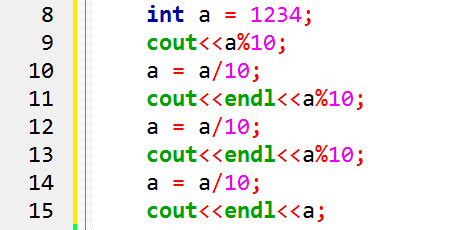
cout<<"\n \t\t Between 25 and 29.9 : Overweight";

cout<<"\n \t\t Greater than 30 : Overweight";

return 0;

}

Working with +, -, \* and / is very obvious. Modulus operator (%) though needs some more explanation. Modulus operator gives the value of remainder once an int is divided by other int. This is very useful operator in C++. This can be very helpful in many programming situations. The following code snippet will help develop more intuition about modulus operator.



**3.2 Operators Precedence and Associativity**

In order to properly evaluate an expression such as 4 + 2 \* 3, we must understand both what the operators do, and the correct order to apply them. The order in which operators are evaluated in a compound expression is called operator precedence. Using normal mathematical precedence rules (which state that multiplication is resolved before addition), we know that the above expression should evaluate as 4 + (2 \* 3) to produce the value 10.

In C++, all operators are assigned a level of precedence. Those with the highest precedence are evaluated first. You can see in the table below that multiplication and division have a higher precedence than addition and subtraction. The compiler uses these levels to determine how to evaluate expressions it encounters.

Thus, 4 + 2 \* 3 evaluates as 4 + (2 \* 3) because multiplication has a higher level of precedence than addition.

If two operators with the same precedence level are adjacent to each other in an expression, the associativity rules tell the compiler whether to evaluate the operators from left to right or from right to left.

For example, in the expression 3 \* 4 / 2, the multiplication and division operators are both precedence level 5. Level 5 has an associativity of left to right, so the expression is resolved from left to right: (3 \* 4) / 2 = 6.

|  |  |  |
| --- | --- | --- |
| **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 sub expressions 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. |

Following examples will help you to understand the idea of precedence and associativity

y = 5 / 2 \* 5 + 3 \* 5 + 7;

cout<<y;

y = 5 \* 5 / 2 + 3 \* 5 + 7;

cout<<y;

Now try the following codes

Code 03

1. #include<iostream>
3. using namespace std;
4. int main()
5. {
6. int number1 = 74, number2 = 82, number3 = 88;
7. double average;
8. average = number1 + number2 + number3 / 3;
9. cout<<average;
10. return 0;
11. }

**Code 04**

1. #include<iostream>
2. using namespace std;
3. int main()
4. {
5. int number1 = 74, number2 = 82, number3 = 88;
6. double average;
7. average = (number1 + number2 + number3) / 3;
8. cout<<average;
9. return 0;
10. }

What did you observe from the output of the above two programs? Try to explain briefly.

Code 03:As we know that division has high precedence level so in this case number3 will be divided by 3 and the result obtained will be added into number1 and number2 to give output of the program

Output=185.33 (Double inp number) or 185(int inp number)

Code 04:In this case as parenthesis are present in numerator and we know that parenthesis has higher precedence than division so numbers inside the parenthesis will add first and then divide by 3 to give output.

Output= 81.33(double input) or 81 (int input)

**3.3 Special Mathematical Operators (Assignment Operators): Increment and Decrement**

Incrementing (adding 1 to) and decrementing (subtracting 1 from) a variable are so common that they have their own operators in C++. There are actually two versions of each operator, a prefix version and a postfix version. Following table lists them

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Form** | **Operation** |
| Prefix increment (pre-increment) | ++ | ++x | Increment x, then evaluate x |
| Prefix decrement (pre-decrement) | -- | --x | Decrement x, then evaluate x |
| Postfix increment (post-increment) | ++ | x++ | Evaluate x, then increment x |
| Postfix decrement (post-decrement) | -- | x-- | Evaluate x, then decrement x |

The prefix increment/decrement operators are very straightforward. The value of x is incremented or decremented, and then x is evaluated.

For example

int x = 5;

int y = ++x; // x is now equal to 6, and 6 is assigned to y

The postfix increment/decrement operators are a little more tricky. The compiler makes a temporary copy of x, increments or decrements the original x (not the copy), and then evaluates the temporary copy of x. The temporary copy of x is then discarded.

int x = 5;

int y = x++; // x is now equal to 6, and 5 is assigned to y

Let’s examine how this last line works in more detail. First, the compiler makes a temporary copy of x that starts with the same value as x (5). Then it increments the original x from 5 to 6. Then the compiler evaluates the temporary copy, which evaluates to 5, and assigns that value to y. Then the temporary copy is discarded.

Consequently, y ends up with the value of 5, and x ends up with the value 6. Here is another example showing the difference between the prefix and postfix versions:

int x = 5, y = 5;

cout << x << " " << y << endl;

cout << ++x << " " << --y << endl; // prefix

cout << x << " " << y << endl;

cout << x++ << " " << y-- << endl; // postfix

cout << x << " " << y << endl;

This produces the output:

5 5

6 4

6 4

6 4

7 3

**Special Mathematical Operators (Assignment Operators): Compound Assignments**

Compound assignment operators modify the current value of a variable by performing an operation on it. They are equivalent to assigning the result of an operation to the first operand: Following table summarizes the compound assignments

|  |  |
| --- | --- |
| **Equation with Compound Assignment** | **Actually means** |
| x +=3.5 | x=x+3.5 |
| x -= 1000 | x=x-1000 |
| x \*= 10 | x=x\*10 |
| x /= 5 | x=x/5 |

Evaluating Expression and Equations with Mixed Data Types

You are now familiar with the idea of precedence and associativity. It is now time to clarify one very important aspects of C++ mathematics, how an expression or equation contacting mixed data types e.g. int and float is evaluated. Consider the equation for example

tempf=tempc\*(9/5)+32;

One may be disguised that there is nothing wrong with the above statement, but the way C++ handle it is really important to consider. The literal 9 when divided by 5 will result in an int value whereas the user might be expecting floating result. In that case the result will be incorrect. This can be corrected by implementing the same expression with floating point literals, like

tempf=tempc\*(9.0/5.0)+32;

Following will also do the job.

tempf=tempc\*(9.0/5)+32; or tempf=tempc\*(9/5.0)+32;

**3.4 Type Casting**

C++ allows to temporarily change the type of a variable for one statement, this idea is called type casting. The idea is explained in the following code.

**Code 05**

1. #include<iostream>
2. #include<cmath>
3. using namespace std;
4. int main ()
5. {
6. float num1 = -9.5;
7. int num2 = 101;
8. cout<<(int)num1;
9. cout<<endl<<(float)num2/10;
10. return 0;
11. }

Line number 10 will be processed by considering num1 as int and not its own type, similarly line 11 will be executed by considering num2 as floating point quantity and not int.

**3.5 Advanced Mathematical Functions <cmath>**

Some very useful and advanced mathematical functions are present in <cmath> library. Which can be included in a program through preprocessor directive #include<cmath>. Following are the few functions which are available in this library.

|  |  |  |
| --- | --- | --- |
| **Category** | **Function** | **Description** |
| Trigonometry | cos | Returns the cosine of an angle of x radians. |
| sin | Returns the sine of an angle of x radians. |
| tan | Returns the tangent of an angle of x radians. |
| acos | **The acos** function computes the principal value of the arc cosine of **x.** A domain error occurs  for arguments not in the range [-I. +I]. |
| asin | The **asin** function computes the principal value of the arc sine of **x.** A domain error occurs  for arguments not in the range [-I, +I]. |
| atan | The atan function returns the arc tangent in the range [[-pi/2, +pi/2]] |
| Exponential and logarithmic function | exp | Returns the base-e exponential function of *x*, which is e raised to the power *x*: ex. |
| log | Returns the natural logarithm of *x*.  If the argument is negative, a domain error occurs. |
| log10 | Returns the common (base-10) logarithm of *x*.  If the argument is negative, a domain error occurs. |
| Power Functions | pow | Returns base raised to the power exponent:  e.g. pow(7.0, 3.0); will find 7 ^ 3 |
| sqrt | Returns the square root of *x*.  If *x* is negative, a domain error occurs: |
| cbrt | Returns the cubic root of *x*. |
| Rounding and Remainder Functions | ceil | Rounds *x* upward, returning the smallest integral value that is not less than *x*. |
| floor | Rounds *x* downward, returning the largest integral value that is not greater than *x*. |
| fmod | Returns the floating-point remainder of *numer*/*denom* |
| trunc | Rounds *x* toward zero, returning the nearest integral value that is not larger in magnitude than *x*. |
| round | Returns the integral value that is nearest to *x*, with halfway cases rounded away from zero. |
| Other Functions | fabs | Returns the absolute value of *x*: |*x*|. |
| abs | Returns the absolute value of *x*: |*x*|. |

Example to use trigonometric functions

Code 06

1. #include<iostream>
2. #include<cmath>
3. using namespace std;
4. int main()
5. {
6. const double pi = 3.141592;
7. double angle = pi/6;
8. cout<<endl<<"\*\*\*\*\*\*\*\* Calculating Trigonometric Ratios \*\*\*\*\*\*\*\*"<<endl;
9. cout<<endl<<"All calculations on Angle "<<angle<<" Radians"<<endl;
10. cout<<endl<<"cos("<<angle<<") "<<"= "<<cos(angle)<<endl;
11. cout<<endl<<"sin("<<angle<<") "<<"= "<<sin(angle)<<endl;
12. cout<<endl<<"tan("<<angle<<") "<<"= "<<tan(angle)<<endl;
13. cout<<endl<<"\*\*\*\*\*\*\*\* Calculations Terminated \*\*\*\*\*\*\*\*"<<endl;
14. return 0;
15. }

Example to use exponential and logarithmic functions

Code 07

1. #include<iostream>
2. #include<cmath>
3. using namespace std;
4. int main()
5. {
6. double num = 10.3;
7. cout<<endl<<"exp("<<num<<") "<<"= "<<exp(num)<<endl;
8. cout<<endl<<"log("<<num<<") "<<"= "<<log(num)<<endl;
9. cout<<endl<<"log10("<<num<<") "<<"= "<<log10(num)<<endl;
10. return 0;
11. }

Example to use power functions

Code 08

1. #include<iostream>
2. #include<cmath>
3. using namespace std;
4. int main()
5. {
6. double num1 = 10.3, num2 = 2.0;
7. cout<<endl<<"pow("<<num1<<","<<num2<<") "<<"= "<<pow(num1,num2)<<endl;
8. cout<<endl<<"sqrt("<<num1<<") "<<"= "<<sqrt(num1)<<endl;
9. cout<<endl<<"cbrt("<<num1<<") "<<"= "<<cbrt(num1)<<endl;
10. return 0;
11. }

Example to use power rounding functions

Code 09

1. #include<iostream>
2. #include<cmath>
3. using namespace std;
4. int main()
5. {
6. double num1 = 2.3,num2 = 3.8,num3 = 5.5,num4 = -2.3,num5 = -3.8,num6 = -5.5;
7. cout<<"value\tround\tfloor\tceil\ttrunc\n";
8. cout<<"-----\t-----\t-----\t----\t-----\n";
9. cout<<num1<<"\t"<<round(num1)<<"\t"<<floor(num1)<<"\t"<<ceil(num1)<<"\t"<<trunc(num1)<<"\n";
10. cout<<num2<<"\t"<<round(num2)<<"\t"<<floor(num2)<<"\t"<<ceil(num2)<<"\t"<<trunc(num2)<<"\n";
11. cout<<num3<<"\t"<<round(num3)<<"\t"<<floor(num3)<<"\t"<<ceil(num3)<<"\t"<<trunc(num3)<<"\n";
12. cout<<num4<<"\t"<<round(num4)<<"\t"<<floor(num4)<<"\t"<<ceil(num4)<<"\t"<<trunc(num4)<<"\n";
13. cout<<num5<<"\t"<<round(num5)<<"\t"<<floor(num5)<<"\t"<<ceil(num5)<<"\t"<<trunc(num5)<<"\n";
14. cout<<num6<<"\t"<<round(num6)<<"\t"<<floor(num6)<<"\t"<<ceil(num6)<<"\t"<<trunc(num6)<<"\n";
15. return 0;
16. }

Exercise

**Task 1:**

Using compound assignment operators, write a program that generates the following output:

|  |
| --- |
| x = 2.5 y = 10  x = 25.0 y = 15  x = 250.0 y = 20  x = 2500.0 y = 25 |

Initialize x as float with value of 2.5 and y as int with value 10. In each successive stage, use \*= operator for x and += operator for y to achieve the desired values.

#include <iostream>

using namespace std;

int main()

{

float x=2.5;

int y=10;

for(int i=1;i<5;i++)//counter runs 4 times

{

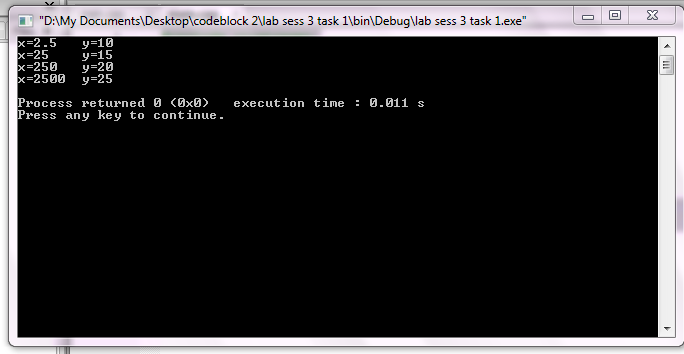
cout<<"x="<<x<<"\ty="<<y<<endl;//initially given values of x and y would print

x=x\*10.0;//after that the values substitute in these formulas and print output

y=y+5;

}

}



**Task 2:**

Write a program that asks the user to enter the length of base and perpendicular of a right angle triangle. Then it determines the length of hypotenuse, angle between base and hypotenuse and angle between hypotenuse and perpendicular. Also find the sine and cosine values of these angles. (For hint refer to basic trigonometry from any mathematics book)

#include <iostream>

#include<cmath>

using namespace std;

int main()

{

double angle1,angle2;

float p,b,h;

cout<<"Enter length of base:"<<endl;

cin>>b;

cout<<"Enter length of perp:"<<endl;

cin>>p;

h = sqrt( p \* p + b \* b );

angle1=acos(b/h);//acos=cos^-1

angle2=asin(p/h);//theta=sin^-1(p/h)

cout<<"length of hyp ="<<h<<endl;

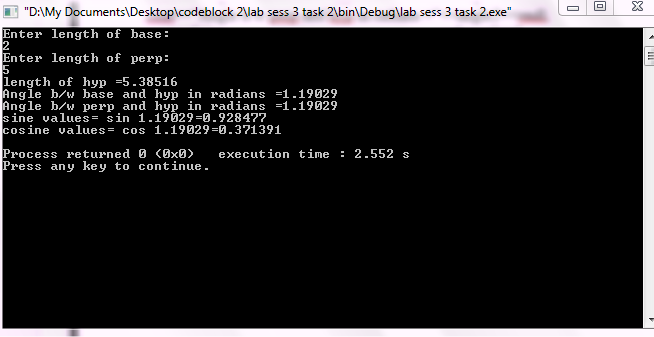
cout<<"Angle b/w base and hyp in radians ="<<angle1<<endl;

cout<<"Angle b/w perp and hyp in radians ="<<angle2<<endl;

cout<<"sine values="<<" sin "<<(angle2)<<"="<<sin(angle2)<<endl;//sin/cos/tan(angle)formula

cout<<"cosine values="<<" cos "<<(angle1)<<"="<<cos(angle1)<<endl;

}



**Task 3:**

Write a program that asks the user to enter coefficients a, b and c of the standard quadratic equation:

ax2+bx+c=0

The program then should compute and display discriminant

**|**b2-4ac**|**

And the roots of equation

Finally, give opinion on how the program could be made more general to different input conditions

#include <iostream>

#include<cmath>

using namespace std;

int main()

{

int a,b,c;

float D,x1,x2;

cout<<"Enter coefficients of a b and c:"<<endl;

cin>>a>>b>>c;

D=abs((b\*b)-(4\*a\*c));

cout<<"discriminant="<<D<<endl;

if(b<4\*a\*c)

{

cout<<"in order to prevent imaginary roots b should be greater or equals to 4ac i.e (b>=4ac)"<<endl;

}

else

{

x1=(-b+(sqrt((b\*b)-(4\*a\*c)))/(2\*a));

x2=(-b-(sqrt((b\*b)-(4\*a\*c)))/(2\*a));

cout<<"Roots of the equation "<<a<<"x^2"<<" + "<<b<<"x"<<" + "<<c<<" are "<<": x1 = "<<x1<<" , "<<" x2 = "<<x2<<endl;

}

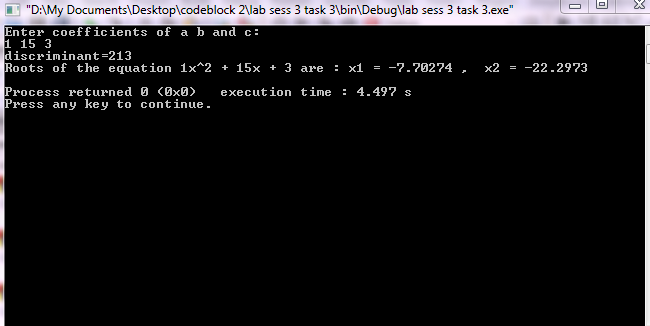
}

As value of discriminant should be positive that’s why we used abs function with it

Now for roots of equation:

When (b>=4ac):

The output is:



When (b<4ac):

In this the roots of the equation will be imaginary

The output is

