



---

# PROGRAMMING FUNDAMENTALS

---

Lab Manual



SESSION: 2020  
SEMESTER: 1ST  
Course Code: CS-142L

## Contents

<b>Lab 01: Problem solving and Algorithms.....</b>	<b>2</b>
<b>Lab 02: Integrated Development Environment (IDE) and Basics of Programming .....</b>	<b>4</b>
<b>Lab 03: Output/ Escape Sequences .....</b>	<b>13</b>
<b>LAB 04: Variables and Data types .....</b>	<b>16</b>
<b>LAB 05: Named Constants and Type Casting .....</b>	<b>25</b>
<b>LAB 06: Control Structures (if/if-else/if-else-if) .....</b>	<b>31</b>
<b>LAB 07: Control Structures (switch Statement/Nested if).....</b>	<b>35</b>
<b>LAB 08: Control Structures (for/while/do-while loop) .....</b>	<b>39</b>
<b>LAB 09: Arrays.....</b>	<b>52</b>
<b>LAB 10: Arrays.....</b>	<b>57</b>
<b>LAB 11: Functions .....</b>	<b>61</b>
<b>LAB 12: Functions .....</b>	<b>70</b>
<b>LAB 13: Searching and Sorting Arrays.....</b>	<b>78</b>
<b>LAB 14: File Handling .....</b>	<b>89</b>
<b>LAB 15: Pointers.....</b>	<b>97</b>
<b>LAB 16: Structures .....</b>	<b>106</b>

CLO1:

Apply appropriate programming techniques to create executable programs to solve well defined problems

CLO2:

Practice collaboratively on large problems and provide their working solutions.

CLO3:

Adhere to plagiarism guidelines

CLO4:

Display well-commented code

## Lab 01: Problem solving and Algorithms

**Objective(s):** Learn about problem solving and algorithms.

**CLOs:** CL01, CLO4

In computing, we focus on the type of problems categorically known as algorithmic problems, where their solutions are expressible in the form of algorithms.

An algorithm a well-defined computational procedure consisting of a set of instructions that takes some value or set of values, as input, and produces some value or set of values, as output. In other word, an algorithm is a procedure that accepts data; manipulate them following the prescribed steps, so as to eventually fill the required unknown with the desired value(s).



Tersely put, an algorithm, a jargon of computer specialists, is simply a procedure. People of different professions have their own form of procedure in their line of work, and they call it different names. A cook, for instance, follows a procedure commonly known as a recipe that converts the ingredients (input) into some culinary dish (output), after a certain number of steps.

An algorithm is a form that embeds the complete logic of the solution. Its formal written version is called a program, or code. Thus, algorithmic problem solving actually comes in two phases: derivation of an algorithm that solves the problem, and conversion of the algorithm into code. The latter, usually known as coding, is comparatively easier, since the logic is already present – it is just a matter of ensuring that the syntax rules of the programming language are adhered to. The first phase is what that stumbles most people, for two main reasons. Firstly, it challenges the mental faculties to search for the right solution, and secondly, it requires the ability to articulate the solution concisely into step- by-step instructions, a skill that is acquired only through lots of practice. Many people are able to make claims like “oh yes, I know how to solve it”, but fall short when it comes to transferring the solution in their head onto paper.

Algorithms and their alter ego, programs, are the software. The machine that runs the programs is the hardware.

**Pseudocode** is a method of describing computer algorithms using a combination of natural language and programming language. It is essentially an intermittent step towards the development of the actual code. It allows the programmer to formulate their thoughts on the organization and sequence of a computer algorithm without the need for actually following the exact coding syntax. Although pseudocode is frequently used there are no set of rules for its exact implementation. In general, here are some rules that are frequently followed when writing pseudocode:

Symbols are used for arithmetic operations (+, -, \*, /, \*\*).

Symbolic names are used to indicate the quantities being processed.

Certain keywords can be used, such as PRINT, WRITE, READ, INPUT, OUTPUT etc.

Indentation should be used to indicate branches and loops of instruction.

### **Examples of Algorithms:**

#### **Write an algorithm to find area of a rectangle**

Step 1: Start

Step 2: get l and b values

Step 3: Calculate  $A = l * b$

Step 4: Display A

Step 5: Stop

#### **Write an algorithm for Calculating area and circumference of circle**

Step 1: Start

Step 2: get r value

Step 3: Calculate  $A = 3.14 * r * r$

Step 4: Calculate  $C = 2.3.14 * r$

Step 5: Display A, C

Step 6: Stop

#### **To check greatest of two numbers**

Step 1: Start

Step 2: get num1 and num2 value

Step 3: check if (a>b) print num1 is greater

Step 4: else num2 is greater

Step 5: Stop

### **Lab Tasks:**

1. Write an algorithm to add two numbers
2. Write an algorithm to find velocity of a moving object
3. Write an algorithm to check positive or negative number
4. Write an algorithm to find volume of cylinder

**Lab 02:****Integrated Development Environment (IDE) and Basics of Programming****Objective(s):**

Understanding of Integrated Development Environment (IDE)

**CLOs:** CL01, CLO4

**Tools(s):**

- PC with Windows 7 Professional or above
- Visual Studio 2010 or above

**HISTORY OF C and C++**

C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization.

**INTRODUCTION (IDE):**

An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. In general, an IDE is a graphical user interface (GUI) - based workbench designed to aid a developer in building software applications with an integrated environment combined with all the required tools at hand.

**Examples of IDEs:****1) Microsoft Visual Studio**

Microsoft Visual Studio is IDE for Windows application development, look no further than to Microsoft's own developer toolset. Visual Studio products cover languages like C++, C# and VB.NET. In addition, you are also able to develop for the Windows x86, Windows RT, and Windows Phone. The latest version of Visual Studio is also designed to be optimized for touch, just in case you happen to be writing code on a Microsoft Surface.

**2) Eclipse**

Eclipse contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. By means of various plugins, Eclipse may also be used to develop applications in other programming languages: Ada, ABAP, C, C++, COBOL, FORTRAN, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, and Erlang.

**3) Code Blocks**

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

#### 4) Turbo C/C++

Turbo C is an Integrated Development Environment and compiler for the C programming language from Borland. First introduced in 1987, it was noted for its integrated development environment, small size, fast compile speed, comprehensive manuals and low price.

#### Installation of Visual Studio

##### STEP 1

Download Setup of Visual Studio 2010 or latest version of Visual Studio 2019

##### STEP 2

Click “install Microsoft Visual Studio 2010” in order to install Visual Studio.

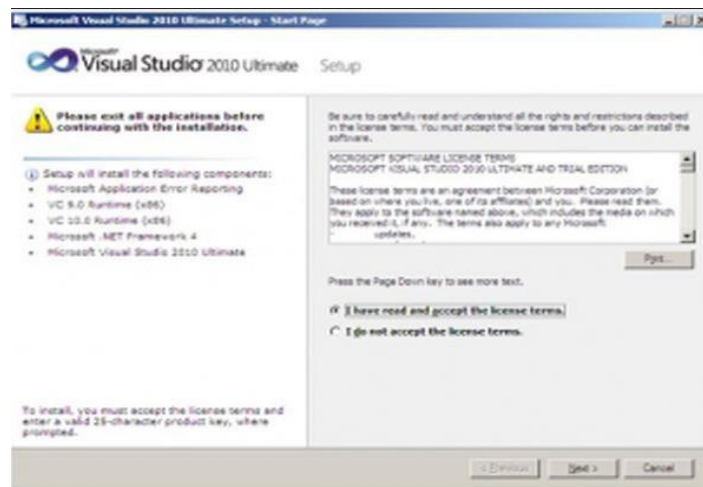


##### STEP 3

Click —Next to start installation

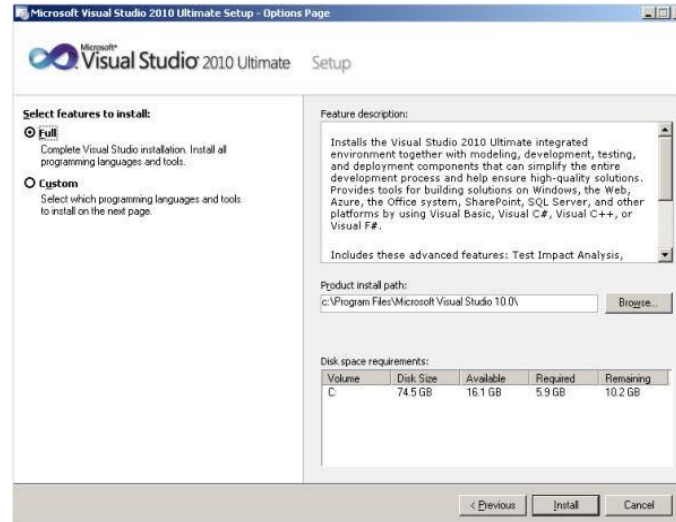
##### STEP 4

After clicking next you will see the following screen. Click the button “I have read and accept the license agreement”



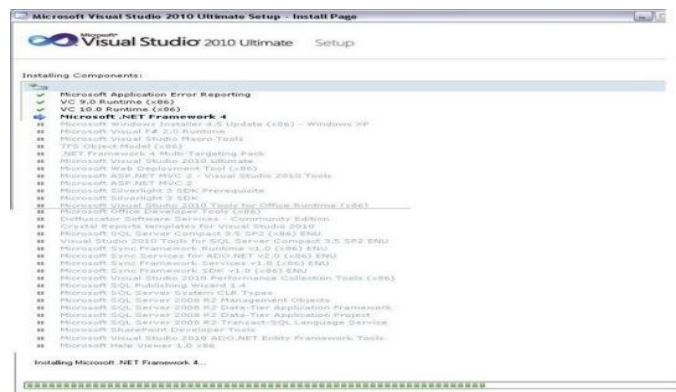
## STEP 5

Select the features you want to install in Visual Studio 2010. You may select Full Installation or Custom Installation for selected components to be added. Also select the location where you want to install Visual Studio.



## STEP 6

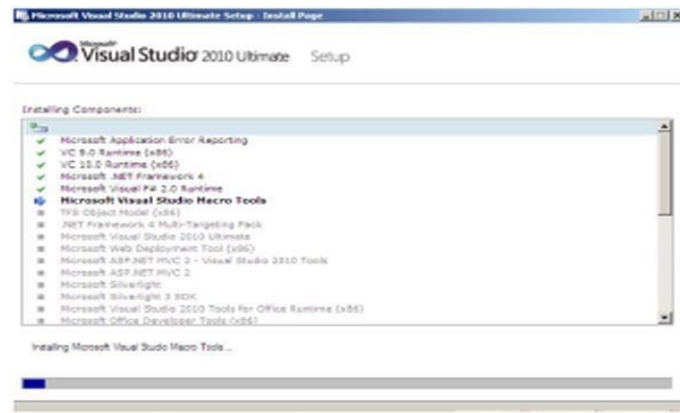
Now the setup will start to install its components.



## STEP 7

Now a window appears which shows the components of Visual Studio being installed.





## STEP 8

Click Finish. The installation is complete now.



## 1) Header Files

A header file is a file which contains C++ function declarations and macro definitions and to be shared between several source files. You request the use of a header file in your program by including it, with the C++ preprocessing directive `#include` like you have seen inclusion of `iostream` header file, which comes along with your compiler.

## 2) Using namespace std;

The namespace creates a declarative region in which various program elements are defined. The using statement informs the compiler that you want to use the `std` namespace. If the following line is not included the `cout` would not have been executed, as `cout` is included in the namespace `std`.

### 3) Main Function

void main() is the first statement executed whenever the C++ program is run. It is the starting point of the program. If main function is not included in the program, the compiler will show an error. void is a data type of function main, it shows the program is not returning any value

**4) cout<<"Hello World";**

This line is a statement. Statements in C++ always end with semi-colon. Statements are always executed in the order they appear. The cout corresponds to a standard output stream. It is used to display the output on the screen. The symbol "<<" refers to an *insertion* operator. Its function is to direct the string constants to cout which displays it onto the screen.

### Compilation and Running a Program

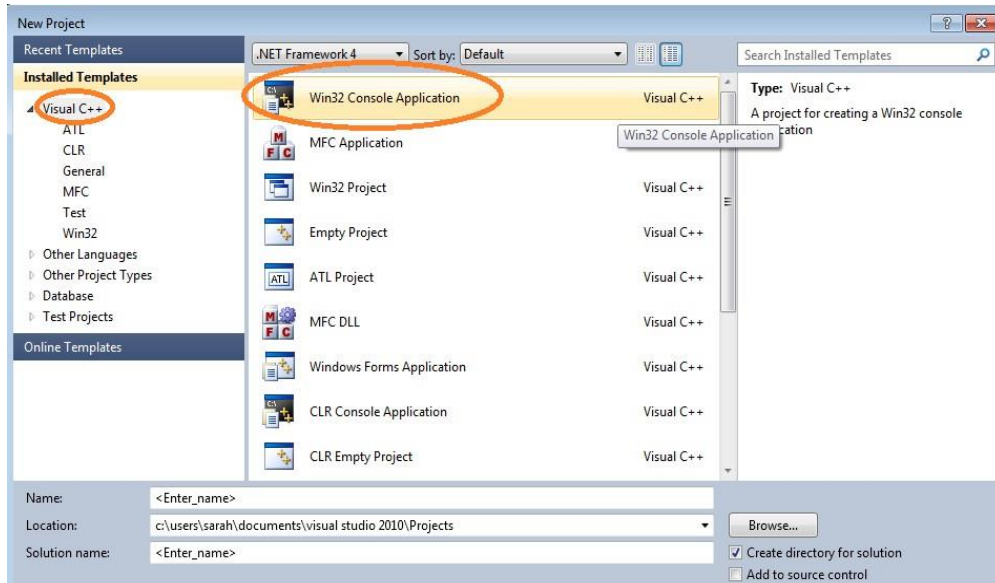
Press Ctrl+ Shift + B to compile. If there is no error the program compiles successfully. Press Ctrl + F5 to run the program.

## Creating a Project in Visual Studio

Follow the below mentioned steps to create a new project in VS.

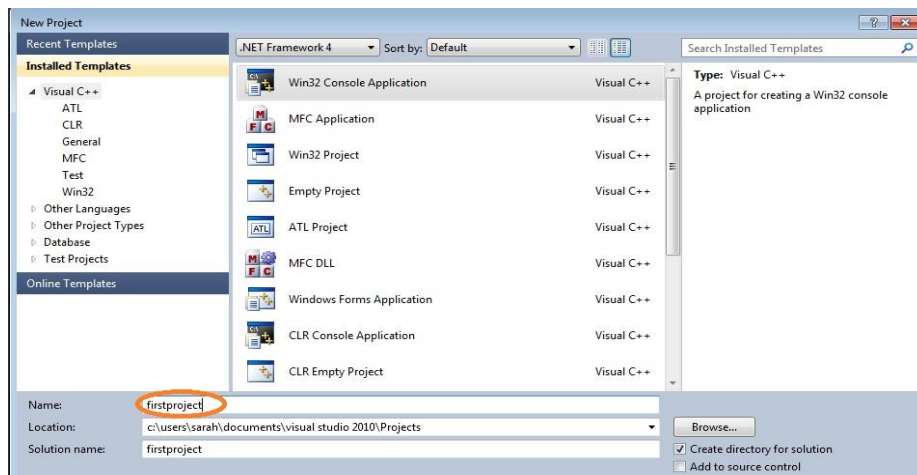
### STEP 1

Click File New Project. From the Installed Templates Select Visual C++, and select Empty Project.



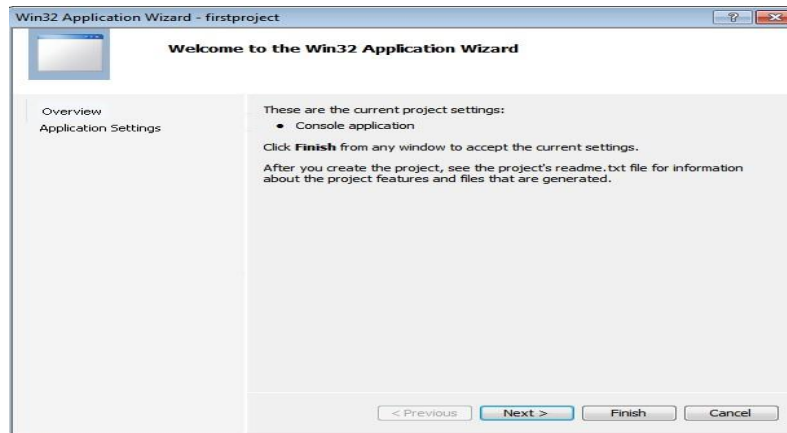
### STEP 2

Name your Project as “firstproject” and click OK.



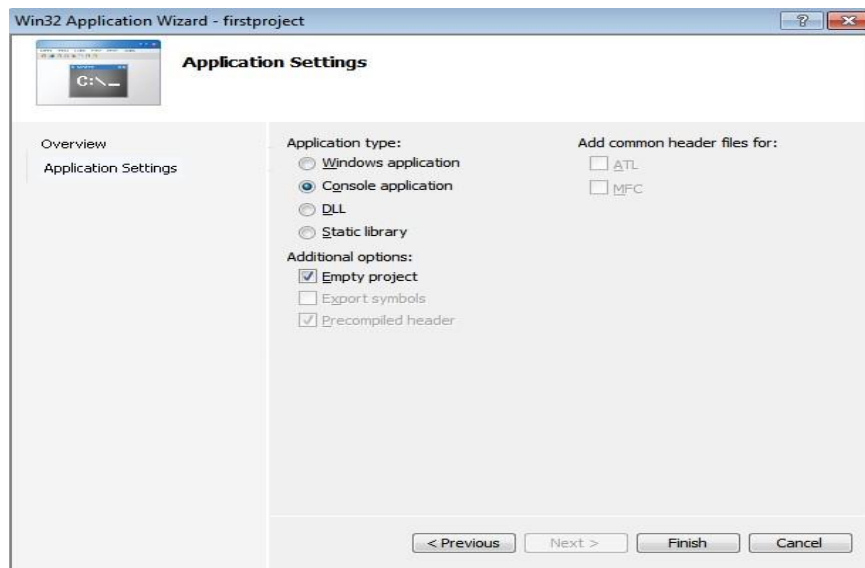
### STEP 3

This will open an Application Wizard. Click Next to continue.



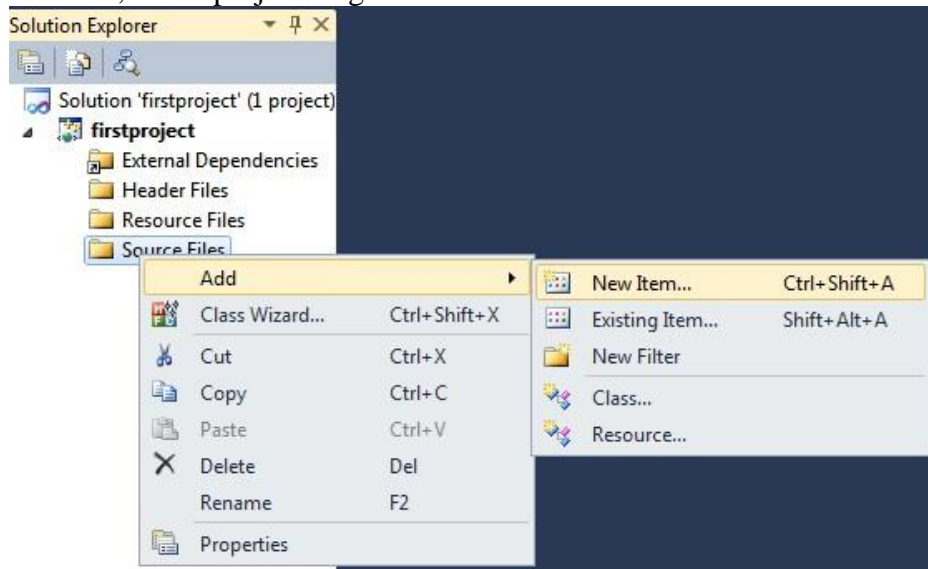
### STEP 4

Now leave —Console Application‖ selected, and in additional options select —Empty Project. Click —Finish.



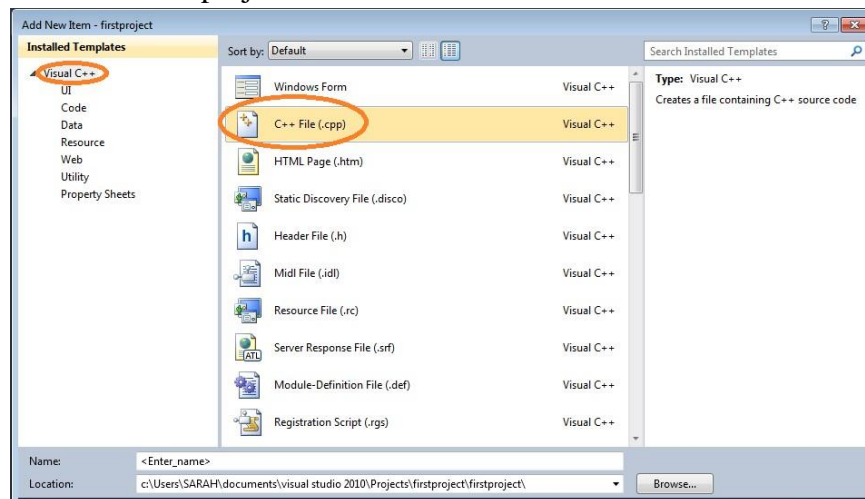
## STEP 5

Add a source file, to the project. Right Click Add New Item



## STEP 6

Select Visual C++ from the —Installed Templates. Select .cpp extension file from the list. Name the source file first project.



## Lab 03: Output/ Escape Sequences

**Objective(s):** Learn about input and output in C++.

**CLOs:** CL01, CLO4

**cout<<**

Statements in C++ always end with semi-colon. Statements are always executed in the order they appear. The cout corresponds to a standard output stream. It is used to display the output on the screen. The symbol "<<" refers to an *insertion* operator. Its function is to direct the string constants to cout which displays it onto the screen.

**Example:**

Write your first C++ Program.

```
// A simple C++ program
#include <iostream>      //Header file
using namespace std;
int main()              // Main function
{
    cout << "Programming is great fun!";
    return 0;
}
```

**OUTPUT**

Programming is great fun!

In the sample program you encountered several sets of special characters. Table below provides a short summary of how they were used.

**Special Characters**

Character	Name	Description
//	Double slash	Marks the beginning of a comment.
#	Pound sign	Marks the beginning of a preprocessor directive.
<>	Opening and closing brackets	Encloses a filename when used with the #include directive.
()	Opening and closing parentheses	Used in naming a function, as in int main()
{ }	Opening and closing braces	Encloses a group of statements, such as the contents of a function.

" "	Opening and closing quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen.
;	Semicolon	Marks the end of a complete programming statement

### Escape Sequences:

Escape sequences are special characters used in control string to modify the format of an output. These specific characters are translated into another character or a sequence of characters that may be difficult to represent directly. For example, you want to put a line break in the output of a C++ statement then you will use “\n” character which is an escape sequence itself.

An escape sequence consists of two or more characters. For all sequences, the first character will be “\” i.e. backslash. The other characters determine the interpretation of escape sequence. For example, “n” of “\n” tells the cursor to move on the next line.

### Common Escape Sequences:

Escape Sequence	Name	Description
\n	Newline	Causes the cursor to go to the next line for subsequent printing.
\t	Horizontal tab	Causes the cursor to skip over to the next tab stop.
\a	Alarm	Causes the computer to beep.
\b	Backspace	Causes the cursor to back up, or move left one position.
\r	Return	Causes the cursor to go to the beginning of the current line, not the next line.
\\	Backslash	Causes a backslash to be printed.
\'	Single quote	Causes a single quotation mark to be printed.
\"	Double quote	Causes a double quotation mark to be printed

### Examples:

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

int main() {

    cout<< "Hey, \"How are you?\"";

    getch();

    return 0;
```

```
}
```

Output

Hey, "How are you?"

**Lab Tasks:**

**Task 1:**

Write a program that displays the following pieces of information, each on a separate line:

Your name

Your address, with city, province and country

Your telephone number

Your university name

Your program

Your Semester

Use only a single cout statement to display all of this information.

**Task 2:**

Write a program to show the following output?

1 4 7

2 5 8

3 6 9

**Task 03:**

Write a program to display

```
  *  
  
*  *  
  
*    *  
  
*  *  
  
  *
```



## LAB 04: Variables and Data types

### Objective(s):

Understanding of Data types, Variables and operations that can be performed using them.

**CLOs:** CLO1, CLO2

### Data types:

Data types portray the amount of memory to be allocated. Most common data types are integer, floating- point, character, strings, bool.

### Built – In Data Types in C++:

Type Name	Bytes	Other Names	Range of Values
Int	4	signed	-2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
__int8	1	char	-128 to 127
unsigned __int8	1	unsigned char	0 to 255
__int16	2	short, short int, signed short int	-32,768 to 32,767
unsigned __int16	2	unsigned short, unsigned short int	0 to 65,535
__int32	4	signed, signed int, int	-2,147,483,648 to 2,147,483,647
unsigned __int32	4	unsigned, unsigned int	0 to 4,294,967,295
__int64	8	long long, signed long long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned __int64	8	unsigned long long	0 to 18,446,744,073,709,551,615
Bool	1	none	false or true
Char	1	none	-128 to 127 by default 0 to 255 when compiled by using /J
signed char	1	none	-128 to 127
unsigned char	1	none	0 to 255
Short	2	short int, signed short int	-32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
Long	4	long int, signed long int	-2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
long long	8	none (but equivalent to __int64)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	none (but equivalent to unsigned __int64)	0 to 18,446,744,073,709,551,615

Enum	varies	none	
Float	4	none	3.4E +/- 38 (7 digits)
Double	8	none	1.7E +/- 308 (15 digits)
long double	same as double	none	Same as double
wchar_t	2	__wchar_t	0 to 65,535

### **Variables and Literals**

Variables represent storage locations in the computer's memory. Literals are constant values that are assigned to variables.

Variables allow you to store and work with data in the computer's memory. They provide an "interface" to RAM. Part of the job of programming is to determine how many variables a program will need and what types of information they will hold.

**Declaration of Variables:**

Declaration of variables means that compiler assigns a memory location for the variable.

Declaration of a variable consists of data type and the variable name.

Data type var\_name;

e.g.

**int number;**

This is called a variable definition. It tells the compiler the variable's name and the type of data it will hold. This line indicates the variable's name is number.

int var, number;

float var, num, sum;

char ch;

**Initialization of variables:**

Initialization of variables assigns an initial value to the variable declared. Variable value is initialized by giving variable a name consisting of equal sign followed by a constant value.

Data type var\_name = value;

e.g: int num=5;

float num=4.5, sum=0;

char ch='A';

**Example 01:**

// This program has a variable.

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int number;
```

```
    number = 5;
```

```
    cout << "The value in number is " << number << endl;
```

```
    return 0;
```

```
}
```

Output:

The value in number is 5

**Example 02:**

// this program has variables of several types of the integer types.

```
#include <iostream>
using namespace std;
int main ()
{
    int checking;
    unsigned int miles;
    long days;

    checking = -20;
    miles = 4276;
    days = 189000;
    cout << "We have made a long journey of " << miles;
    cout << " miles.\n";
    cout << "Our checking account balance is " << checking;
    cout << "\nAbout " << days << " days ago Columbus ";
    cout << "stood on this spot.\n";
    return 0;
}
```

Program Output:

We have made a long journey of 4276 miles.

Our checking account balance is -20

About 189000 days ago Columbus stood on this spot.

**OPERATORS**

Operators are in C++ to perform operations on variables and constants. Operators in C++ are:

**Arithmetic Operators**

Basic arithmetic operators in C++ are: \*, /, -, +, ++, -- are the arithmetic operators.

Example

Suppose X and Y are two variables with X=5, Y=10

OPERATORS	DESCRIPTION	EXAMPLE
+	Adds both operands	X + Y = 15
-	Subtracts both operands	Y - X = 5
*	Multiply both operands	X * Y = 150
/	Divide both operands	Y / X = 2
%	Remainder after division of numbers	Y / X = 0

## Assignment Operators:

Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

### Example:

```
int x = 10;
```

## Relational Operators

There are following relational operators supported by C++ language

Assume variable A holds 10 and variable B holds 20, then

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is 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.	(A >= B) is not 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.	(A <= B) is true.
----	--	-------------------

### Logical Operators:

There are following logical operators supported by C++ language.

Assume variable A holds 1 and variable B holds 0, then see examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A    B) is 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.	!(A && B) is true.

### Compound Operators

They modify the variable by performing an operation on it.

Example:

```
#include <iostream>
using namespace std;
void main(){
    int a=0,b=1,c=2,d=4,e=8;
    a+=2;
    cout<<"Value of a is "<<a;
    b-=8;
    cout<<"Value of b is "<<b;
    c*=4;
    cout<<"Value of c is "<<c;
    d/=2;
    cout<<"Value of d is "<<d;
    e%=4;
```

```

        cout<<"Value of e is "<<e;
    }

```

**Output:**

Value of a is 2

Value of b is -7

Value of c is 8

Value of d is 2

Value of e is 0

**Input:**

**cin>>**

The cin object in C++ is an object of class istream. It is used to accept the input from the standard input device i.e. keyboard. The cin corresponds to a standard input stream. The symbol ">>" refers to an *extraction* operator

**Examples:**

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int x;
```

```
    /* For single input */
```

```
    cout << "Enter a number: ";
```

```
    cin >> x;
```

```
    cout << "X = " << x;
```

```
    return 0;
```

```
}
```

```
#include <iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
void main()
```

```
{
```

```
    string exampleString;
```

```
    cout << "Enter string: ";
```

```
    cin >> exampleString; // assign input to exampleString
```

```
    cout << "You entered: " << exampleString << endl; // output exampleString
```

```
}
```

## Lab Tasks:

### Task 1

Suppose there are 7.48 gallons in cubic feet. Write a program that ask user to enter the number of gallons and then display the equivalent in cubic feet.

### Task 2

Write a program that gets 2 integers input from user and store them in variables. Do the five basic Arithmetic Operations (+, -, \*, /, %) of the two numbers. Print the results of operations as below.

```
Enter two integer numbers: 10 25
10 + 25 = 35
10 - 25 = -15
10 * 25 = 250
10 / 25 = 0.4
10 % 25 = 10
```

### Task 3

Write a program that prompt user to input course name, obtained marks and total marks.

Calculate the percentage using the below formula

$$\text{marks\_percentage} = (\text{marks\_obtained} / \text{total}) * 100$$

and display the results as follows.

```
Enter Course name : Fundaments_Programming
Obtained Marks : 93
Total Marks : 100

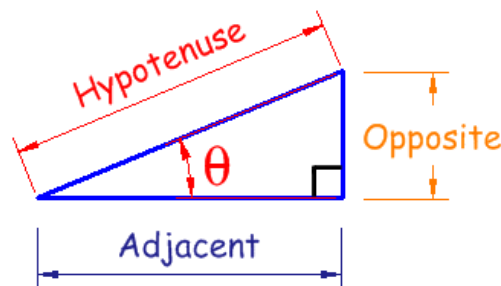
In Fundaments_Programming Course. You have secured %93
```

### Task 4

The distance between two cities (in km.) is input through the keyboard. Write a program to convert and print this distance in meters, feet, inches and centimeters.

### Task 5

Hypotenuse refers to the side opposite the right angle in a right-angled triangle (as shown in the diagram below).



Area of this right-angled triangle can also be calculated using the formula:

$$a = \left(\frac{1}{2}\right) * x * y$$

triangle can also be calculated using the following formula:



**Note that:**

**h= hypotenuse**

**x= adjacent**

Write a C++ program that prompt user to enter value of X and Y. You have to calculate the value of Area (a).

#### **Task 6**

Write a program that prompts the user to enter the weight of a person in kilograms and outputs the equivalent weight in pounds. Output both the weights rounded to two decimal places.

## LAB 05: Named Constants and Type Casting

**CLOs:** CL01, CLO4

### Named constants

A named constant is like a variable, but its content is read-only, and cannot be changed while the program is running. Here is a definition of a named constant:

```
const double INTEREST_RATE = 0.129;
```

It looks just like a regular variable definition except that the word `const` appears before the data type name, and the name of the variable is written in all uppercase characters. The key word `const` is a qualifier that tells the compiler to make the variable read-only. Its value will remain constant throughout the program's execution. It is not required that the variable name be written in all uppercase characters, but many programmers prefer to write them this way so they are easily distinguishable from regular variable names.

### Example:

```
// This program calculates the area of a circle.
// The formula for the area of a circle is PI times
// The radius squared. PI is 3.14159.
#include <iostream>
#include <math.h> // needed for pow function
using namespace std;
int main()
{
    const double PI = 3.14159;
    double area, radius;
    cout << "This program calculates the area of a circle.\n";
    cout << "What is the radius of the circle? ";
    cin >> radius;
    area = PI * pow(radius, 2.0);
    cout << "The area is " << area << endl;
    return 0;
}
```

### Output:

```
This program calculates the area of a circle.
What is the radius of the circle? 2
The area is 12.56
```

### #define preprocessor directive

The #define preprocessor directive creates symbolic constants. The symbolic constant is called a macro and the general form of the directive is –

```
#define macro-name replacement-text
```

When this line appears in a file, all subsequent occurrences of macro in that file will be replaced by replacement-text before the program is compiled. For example –

```
#include <iostream>
```

```
using namespace std;
```

```
#define PI 3.14159
```

```
int main () {
```

```
    cout << "Value of PI :" << PI << endl;
```

```
    return 0;
```

```
}
```

### Formatting output:

The same data can be printed or displayed in several different ways. For example, all of the following numbers have the same value, although they look different:

720

720.0

720.000000000

7.2e+2

+720.0

The way a value is printed is called its formatting. The *cout* object has a standard way of formatting variables of each data type. Sometimes, however, you need more control over the way data is displayed. For formatting manipulator functions we use *<iomanip>* header file in C++ program. There are different types of manipulator functions:

setw

setprecision

fixed

showpoint

left and right

setfill

**Table 3-12**

Stream Manipulator	Description
setw( <i>n</i> )	Establishes a print field of <i>n</i> spaces.
fixed	Displays floating-point numbers in fixed point notation.
showpoint	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
setprecision( <i>n</i> )	Sets the precision of floating-point numbers.
left	Causes subsequent output to be left justified.
right	Causes subsequent output to be right justified.

**Example:**

```
include <iostream>
#include <iomanip>

using namespace std;

void main( )
{
    double no=100.56;
    cout<<setw(10)<<setfill('*')<<"R"<<endl;
    cout<<fixed<<setprecision(8)<<no <<endl;
    system("pause");
}
```

Output:

```
*****R
100.56000000
```

**Overflow and underflow:**

Because data types do have a set minimum to maximum range and a set maximum precision, you cannot represent or store every possible number in standard computer variables.

**Overflow**

Overflow is the situation where you try to store a number that exceeds the value range for the data type. When you try to store too large of a positive or negative number, the binary representation of the number (remember that all values are stored as a 0 and 1 pattern) is corrupted and you get a meaningless or erroneous result.

**Underflow**

Underflow occurs in floating point numbers and is the situation where in numbers very close to zero; there are not enough significant digits to represent the number exactly.

The example below includes two macro constants, INT\_MAX from climits and DBL\_MIN from cfloat.

INT\_MAX: maximum value for an object of type int

DBL\_MIN: minimum representable floating point number for an object of type double

The program

```
#include <climits> // Added to access macro constants
#include <cfloat>
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int maxInt = INT_MAX; // Initialize as largest possible integer
    int biggerThanMax = maxInt * 2; // make too big of an integer
    cout << "maxInt: " << maxInt << endl;
    cout << "biggerThanMax: " << biggerThanMax << endl;

    double closestToZeroDouble = DBL_MIN; // Initialize as smallest possible
    double
    double tooCloseToZeroDouble = closestToZeroDouble / 10.0; // shift decimal
    point closer to 0 (1 to left)

    cout << endl; // insert a blank line
    cout << setprecision(30); // adjusting number of digits displayed

    cout << "closestToZeroDouble: " << closestToZeroDouble << endl;
    cout << "tooCloseToZeroDouble: " << tooCloseToZeroDouble << endl;

    return 0;
}
```

The output

The screenshot shows the output of the program in a terminal window. The output is as follows:

```
maxInt: 2147483647
biggerThanMax: -2
closestToZeroDouble: 2.2250738585072014e-308
tooCloseToZeroDouble: 2.2250738585072034e-309
```

Two annotations are present:

- An arrow points from the text box "I exceeded max so I get garbage. It will not always be a negative number but it will always be incorrect." to the value `-2` in the output.
- An arrow points from the text box "The underflow difference is subtle. If there is no underflow, I should get the same digits to the right of the decimal point with just a different exponent. These differ (ending with 72014 vs. 7203) because of the underflow." to the difference in the last few digits of the exponents in the double outputs.

## Type Casting:

Typecasting is making a variable of one type, such as an int, act like another type, a char, for one single operation. To typecast something, simply put the type of variable you want the actual variable to act as inside parentheses in front of the actual variable. (char)a will make 'a' function as a char.

```
#include <iostream>
using namespace std;
int main()
{
    cout<< (char)65 <<"\n";
    // The (char) is a typecast, telling the computer to interpret the 65 as a
    // character, not as a number. It is going to give the character output of
    // the equivalent of the number 65 (It should be the letter A for ASCII).
    cin.get();
}
```

One use for typecasting for is when you want to use the ASCII characters. For example, what if you want to create your own chart of all 128 ASCII characters. To do this, you will need to use to typecast to allow you to print out the integer as its character equivalent.

```
using namespace std;
int main()
{
    for ( int x = 0; x < 128; x++ ) {
        cout<< x <<" ". << (char)x <<" ";
        //Note the use of the int version of x to
        // output a number and the use of (char) to
        // typecast the x into a character
        // which outputs the ASCII character that
        // corresponds to the current number
    }
    cin.get();
}
```

This is more like a function call than a cast as the type to be cast to is like the name of the function and the value to be cast is like the argument to the function. Next is the named cast, of which there are four:

```
int main()
{
    cout<< static_cast<char> ( 65 ) <<"\n";
    cin.get();
}
```

static\_cast is similar in function to the other casts described above, but the name makes it easier to spot and less tempting to use since it tends to be ugly. Typecasting should be avoided whenever possible. The other three types of named casts are const\_cast, reinterpret\_cast, and dynamic\_cast. They are of no use to us at this time.

## **Lab Tasks:**

### **Task 1**

With the help of manipulator functions write the program that produces the following output

\*\*\*\*\*9997

0.100

1.000e-001

### **Task 2**

Write a program that converts Celsius temperatures to Fahrenheit temperatures. The formula is:

$$F = (9/5) * C + 32$$

F is the Fahrenheit temperature and C is the Celsius temperature.

### **Task 3**

Write a program that prompts the user to enter the weight of a person in kilograms and outputs the equivalent weight in pounds. Output both the weights rounded to two decimal places. (Note that 1 kilogram is equal to 2.2pounds.) Format your output with two decimal places.

### **Task 4**

Write a program that ask user to enter the angle in degrees. Convert the angle in radians and display the output. Keep the value of PI constant.

$$\text{Radians} = \text{Angle} * (\text{PI}/180)$$

## LAB 06: Control Structures (if/if-else/if-else-if)

**Objective(s):** Upon completion of this lab session, learners will be able to:

**CLOs:** CL01, CLO4

Apply if –else decision statements in C++

Understand basic and relational operators used in C++

### Introduction:

A program is usually not limited to a linear sequence of instructions. Normally, statements in a program execute one after the other in the order in which they're written. This is called sequential execution. Various C++ statements we'll soon discuss enable you to specify that *the next statement to execute may be other than the next one in sequence*. This is called transfer of control. C++ provides control structures that serve to specify what has to be done by our program, when and under which circumstances. The instructions in the program can be organized in three kinds of control structures.

If Statement

If/else Statement

Else if

If Statement:

The if statement can cause other statements to execute only under certain conditions.

General format of the if statement

:

```
if (expression)
    statement;
```

### Example 1:

// This program averages 3 test scores.

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
int main() {
```

```
    int score1, score2, score3;
```

```
    double average;
```

```
    cout << "Enter 3 test scores and I will average them: ";
```

```
    cin >> score1 >> score2 >> score3;
```

```
    average = (score1 + score2 + score3) / 3.0;
```

```
    cout << fixed << setprecision(1);
```

```
    cout << "Your average is " << average << endl;
```



```

    if (average == 100)
    {
        cout << "Congratulations! ";
        cout << "That's a perfect score!\n";
    }
    return 0;
}

```

### Program Output

Program Output with Example Input Shown in Bold

Enter 3 test scores and I will average them: 80 90 70 [Enter]

Your average is 80.0

Program Output with Different Example Input Shown in Bold

Enter 3 test scores and I will average them: 100 100 100 [Enter]

Your average is 100.0

Congratulations! That's a high score!

### If / else statement:

The if/else statement will execute one group of statements if the expression is true, or another group of statements if the expression is false. The if/else statement is an expansion of if statement. Here is its format:

```

if (expression)
    statement or block
else
    statement or block

```

Example:

// this program uses an if/else if statement to assign a  
// letter grade (A, B, C, D, or F) to a numeric test score.

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int testScore;
```

```
    char grade;
```

```
    cout << "Enter your numeric test score and I will\n";
```

```
    cout << "tell you the letter grade you earned: ";
```

```
    cin >> testScore;
```

```
    if (testScore < 60)
```

```
        grade = 'F';
```

```
    else if (testScore < 70)
```

```
        grade = 'D';
```

```
    else if (testScore < 80)
```

```
        grade = 'C';
```

```
    else if (testScore < 90)
```

```
        grade = 'B';
```

```
    else if (testScore <= 100)
        grade = 'A';
    cout << "Your grade is " << grade << ".\n";
    return 0;
}
```

Program Output:

Enter your numeric test score and I will tell you the letter grade you earned: 88[Enter] Your grade is B.

## Lab

## Tasks:

### Task 1

Write a C++ Program that read an alphabet (e.g. a,b,c,d,...z) and display whether the input alphabet is a vowel (i.e. a, e, i, o, u) or consonant.

### Task 2

Write a program that asks the user to enter two numbers. The program should use the conditional operator to determine which number is the smaller and which is the larger.

### Task 3

Write a C++ Program that read an integer input in between (1 to 12) and store it month\_of\_year. Print the corresponding month of year. Use else if statement.

*Example: Input is 4... Print "April"*

### Task 4

(Body Mass Index Calculator)

The formulas for calculating, BMI is,

$$BMI = \text{weight In Pounds} \times 703 / (\text{height Inches} \times \text{height Inches})$$

Create a BMI calculator application that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's body mass index. The user can evaluate his/her BMI:

**Values:**

Underweight	Less 18.5
Normal	Between 18.5 and 24.9
Overweight	Between 25 and 29.9
Obese	30 or greater

### Task 5

Write a C++ Menu driven program that allows a user to enter 2 numbers and then choose between findings the sum, subtraction, division, multiplication or average. Use else if statement to determine what action to take.

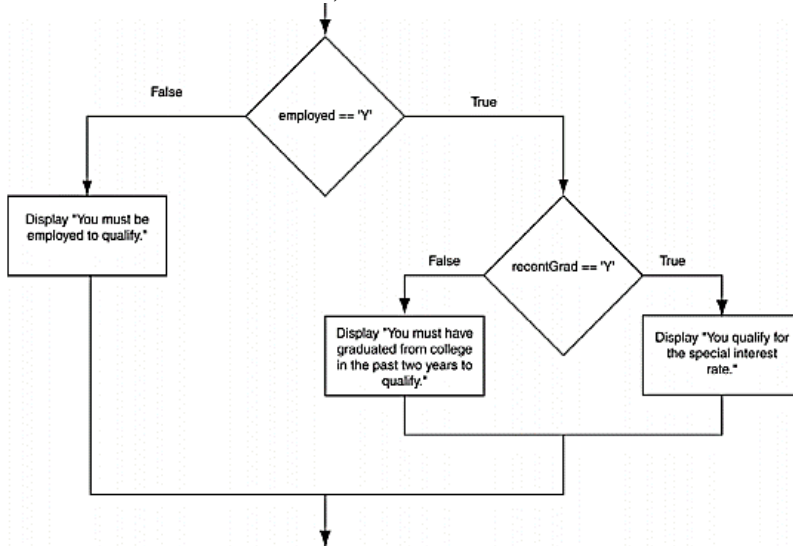
## LAB 07: Control Structures (switch Statement/Nested if)

**Objective(s):** Upon completion of this lab session, learners will be able to:

**CLOs:** CLO1, CLO4

**Nested if Statement:**

Test more than one condition, and if statement can be nested inside another if statement.



Example:

// This program demonstrates a nested if statement.

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    char employed, recentGrad;
```

```
    cout << "Answer the following questions\n";
```

```
    cout << "with either Y for Yes or ";
```

```
    cout << "N for No.\n";
```

```
    cout << "Are you employed? ";
```

```
    cin >> employed;
```

```
    cout << "Have you graduated from college ";
```

```
    cout << "in the past two years? ";
```

```
    cin >> recentGrad;
```

```
    if (employed == 'Y')
```

```
    {
```

```
        if (recentGrad == 'Y') {
```

```
            cout << "You qualify for the special ";
```

```
            cout << "interest rate.\n";
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

### Program Output

Answer the following questions with either Y for Yes or N for No.

Are you employed? Y[Enter]

Have you graduated from college in the past two years? Y[Enter]

You qualify for the special interest rate.

### Program Output

Answer the following questions with either Y for Yes or N for No.

Are you employed? Y[Enter]

Have you graduated from college in the past two years? N[Enter]

### Switch Statement:

The switch statement lets the value of a variable or expression determines where the program will branch.

### Syntax:

Switch (*IntegerExpression*)

```
{
    case ConstantExpression:
        // place one or more
        // statements here
    case ConstantExpression:
        // place one or more
        // statements here
        // case statements may be repeated as many
        // times as necessary
    default:
        // place one or more
        // statements here
}
```

### Example

```
#include <iostream>
using namespace std;
void main (){
    // local variable declaration:
    char grade;
    cout<<"Enter the grade of students ";
    cin>>grade;
    switch(grade){
    case 'A' :
        cout <<"Excellent!\n";
        break;
    case 'B' :
    case 'C' :
        cout << "Well done\n";
        break;
```

```

        case 'D' :
            cout << "You passed\n";
            break;
        case 'F' :
            cout << "Better try again\n";
            break;
        default :
            cout << "Invalid grade\n";
    }
}

```

## OUTPUT

### a) Case “A”

If the user entered letter grade A. The compiler executes case A statements.

```

Enter the grade of students A
Excellent!
Press any key to continue . . .

```

### b) Case “B”

If the user entered letter grade B. The compiler executes case B statements.

```

Enter the grade of students B
Well done
Press any key to continue . . . _

```

### c) Case “C”

If the user entered letter grade C. The compiler executes case C statements.

```

Enter the grade of students C
Well done
Press any key to continue . . . _

```

### d) Case “D”

If the user entered letter grade D. The compiler executes case D statements.

```

Enter the grade of students D
You passed
Press any key to continue . . .

```

### e) Case „F”

If the user entered grade F. The compiler executes case F statements.

```

Enter the grade of students F
Better try again
Press any key to continue . . .

```

f) Default Case:

If any other letter grade is entered. The default statement is executed.

```
Enter the grade of students L
Invalid grade
Press any key to continue . . .
```

Without the break statement, the program “falls through” all of the statements below the one with the matching case expression.

**Lab Tasks:**

**Task 1**

Attempt Task 1 and Task 3 from previous lab using switch statements. (Use symbols as user input operations)

**Task 2**

Serendipity Booksellers has a book club that awards points to its customers based on the number of books purchased each month. (Use Switch Statement)

The points are awarded as follows:

- If a customer purchases 0 books, he or she earns 0 points.
- If a customer purchases 1 book, he or she earns 5 points.
- If a customer purchases 2 books, he or she earns 15 points.
- If a customer purchases 3 books, he or she earns 30 points.
- If a customer purchases 4 or more books, he or she earns 60 points.

Write a program that asks the user to enter the number of books that he or she has purchased this month and then displays the number of points awarded.

## LAB 08: Control Structures (for/while/do-while loop)

**Objective(s):** Upon completion of this lab session, learners will be able to:

**CLOs:** CLO1, CLO4

### The increment and decrement operators

++ and -- are operators that add and subtract 1 from their operands. To *increment* a value means to increase it by one, and to *decrement* a value means to decrease it by one. Both of the following statements increment the variable num:

```
num = num + 1;
```

```
num += 1;
```

And num is decremented in both of the following statements:

```
num = num - 1;
```

```
num -= 1;
```

C++ provides a set of simple unary operators designed just for incrementing and decrementing variables. The increment operator is ++ and the decrement operator is --. The following statement uses the ++ operator to increment num: `num++`; And the following statement decrements num: `num--`;

### Example:

```
// This program demonstrates the ++ and -- operators.
#include <iostream>
using namespace std;
int main()
{
    int num=4;
    // Display the value in num.
    cout << "The variable num is " << num << endl;
    cout << "I will now increment num.\n\n";
    // Use postfix ++ to increment num.
    num++;
    cout << "Now the variable num is " << num << endl;
    cout << "I will increment num again.\n\n";
    // Use prefix ++ to increment num.
    ++num;
    cout << "Now the variable num is " << num << endl;
    cout << "I will now decrement num.\n\n";

    // Use postfix -- to decrement num.
    num--;
    cout << "Now the variable num is " << num << endl;
    cout << "I will decrement num again.\n\n";
}
```



```
}
```

**Program Output:**

The variable num is 4  
I will now increment num.  
Now the variable num is 5  
I will increment num again.  
Now the variable num is 6  
I will now decrement num.  
Now the variable num is 5  
I will decrement num again.  
Now the variable num is 4

**The Difference between Postfix and Prefix Modes:**

It doesn't matter if the increment or decrement operator is used in postfix or prefix mode. The difference is important, however, when these operators are used in statements that do more than just incrementing or decrementing. For example, look at the following lines:

```
num = 4;
```

```
cout << num++;
```

This cout statement is doing two things: (1) displaying the value of num, and (2) incrementing num. But which happens first? cout will display a different value if num is incremented first than if num is incremented last. The answer depends on the mode of the increment operator. Postfix mode causes the increment to happen after the value of the variable is used in the expression. In the example, cout will display 4, then num will be incremented to 5. Prefix mode, however, causes the increment to happen first. In the following statements, num will be incremented to 5, then cout will display 5:

```
num = 4;
```

```
cout << ++num;
```

**Loops:**

A loop is part of a program that repeats. A *loop* is a control structure that causes a statement or group of statements to repeat. C++ has three looping control structures: the while loop, the do-while loop, and for loop. The difference between these structures is how they control the repetition.

**The while Loop:**

The while loop has two important parts: (1) an expression that is tested for a true or false value, and (2) a statement or blocks that is repeated as long as the expression is true.

Here is the general format of the while loop:

Syntax:

```
while (expression)
{
    statement;
    statement;
    // Place as many statements here
    // as necessary.
}
```

**Example:**

// This program

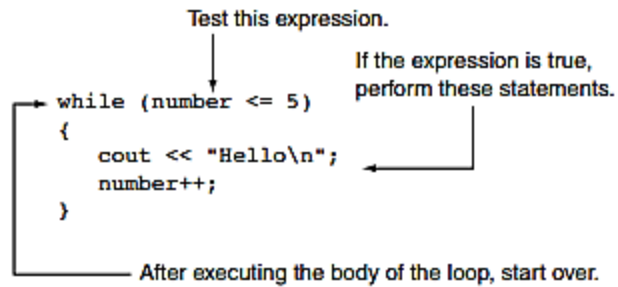
demonstrates a simple while loop.

```
#include <iostream>
using namespace std;
int main()
{
    int number = 1;
    while (number <= 5) {

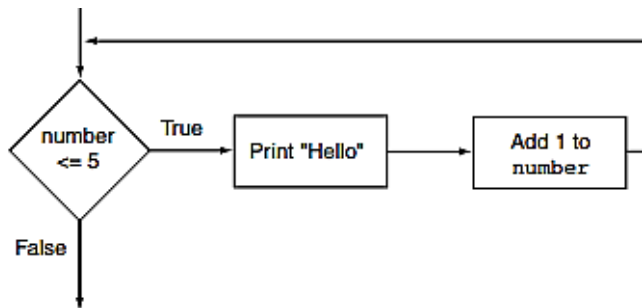
        cout << "Hello\n";
        number++;
    }
    cout << "That's all!\n";
    return 0;
}
```

**Program Output:**

```
Hello
Hello
Hello
Hello
Hello
That's all!
```



Flow diagram:



In this example, the number variable is referred to as the *loop control variable* because it controls the number of times that the loop iterates.

## Counters:

A counter is a variable that is regularly incremented or decremented each time a loop iterates. Sometimes it's important for a program to control or keep track of the number of iterations a loop performs. For example, Program in example displays a table consisting of the numbers 1 through 10 and their squares, so its loop must iterate 10 times.

## Example:

```
// This program displays the numbers 1 through 10 and
// their squares.
#include <iostream>
using namespace std;
int main()
{
    int num =1; // initializing the counter
    cout << "Number Squared\n";

    cout << "-----\n";
    while (num <= 10){
        cout << num << "\t\t" << (num * num) << endl;
        num++; //Increment the counter.
        num++; //Increment the counter.
    }
    return 0;
}
```

## Program Output:

Program Output

Number Squared

-----

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64

## The do-while Loop

The do-while loop is a posttest loop, which means its expression, is tested after each iteration.

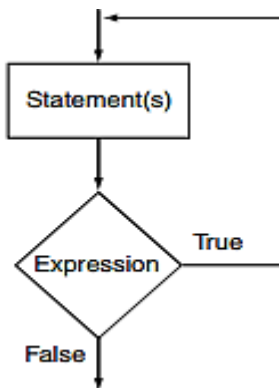
Syntax:

```
do
    statement;
while (expression);
```

Here is the format of the do-while loop when the body of the loop contains multiple statements:

```
do
{
    statement;
    statement;
    // Place as many statements here
    // as necessary.
} while (expression);
```

Flow diagram:



**Example:**

```
// This program averages 3 test scores. It repeats as
// many times as the user wishes.
#include <iostream>
using namespace std;

int main()
{
    int score1, score2, score3; // Three scores
    double average; // Average score

    char again; // To hold Y or N input

    do{
        // get three scores

        Cout<<"enter 3 scores and I will avg them";

        Cin>>score1>>socre 2>>score3;
        /calculate and display the avg.

        average = (score1 + score2 + score3) / 3.0;
        cout << "The average is " << average << ".\n";

        // Does the user want to average another set?

        cin >> again;
    }

    while (again == 'Y' || again == 'y');

    return 0;
}
```

**Program Output:**

```
Enter 3 scores and I will average them: 80 90 70 [Enter]
The average is 80.
Do you want to average another set? (Y/N) y [Enter]
Enter 3 scores and I will average them: 60 75 88 [Enter]
The average is 74.3333.
Do you want to average another set? (Y/N) n [Enter]
```

## The for Loop:

The for loop is ideal for performing a known number of iterations. A count-controlled loop must possess three elements:

It must initialize a counter variable to a starting value.

It must test the counter variable by comparing it to a maximum value. When the counter variable reaches its maximum value, the loop terminates.

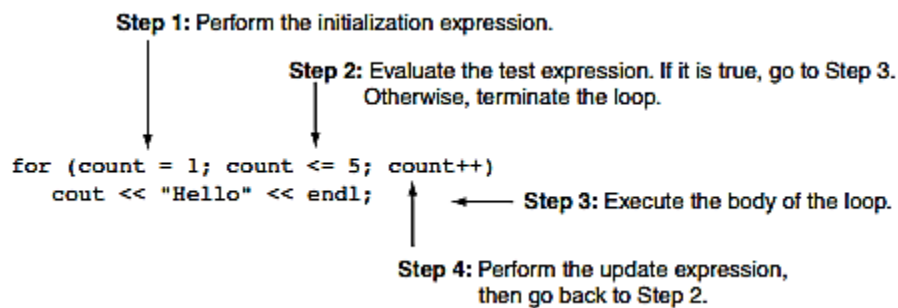
It must update the counter variable during each iteration. This is usually done by incrementing the variable.

Here is the format of the for loop when it is used to repeat a single statement:

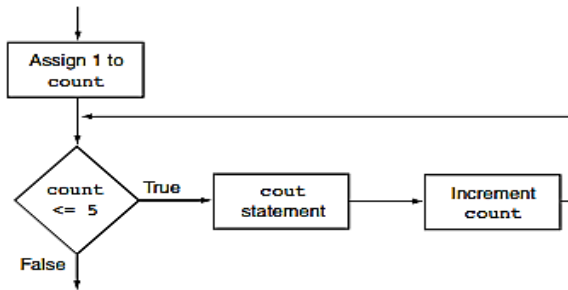
```
for (initialization; test;  
update)
```

The format of the for loop when it is used to repeat a block is

```
for (initialization; test; update)  
{  
statement;  
statement;  
// Place as many statements here  
// as necessary.  
}
```



Flow Diagram:



**Example:**

program displays  
1 through 10 and  
squares.

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

```
int main()
{
    int num;
    cout << "Number Squared\n";
    cout << "-----\n";
    for (num = 1; num <= 10; num++)
        cout << num << "\t\t" << (num * num) << endl;
    return 0;
}
```

// This  
the numbers  
// their

**Nested Loop:**

A loop that is inside another loop is called a *nested loop*.

Example:

// This program averages test scores. It asks the user for the  
// number of students and the number of test scores per student.

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

```
Int main(){

    int numStudents, // Number of students
    numTests; // Number of tests per student

    double total, // Accumulator for total scores
    double total, // Accumulator for total scores
```



```

    average; // Average test score
    // Set up numeric output formatting.

    cout << fixed << showpoint << setprecision(1);
    // Get the number of students.

    cout << "This program averages test scores.\n";

    cout << "For how many students do you have scores? ";
    cin >> numStudents;

    // Get the number of test scores per student.
    cout << "How many test scores does each student have? ";

    cin >> numTests;

    // Determine each student's average score.
    for (int student = 1; student <= numStudents; student++)
    {
        total = 0; // Initialize the accumulator.

        for (int test = 1; test <= numTests; test++)
        {
            double score;

            cout << "Enter score " << test << " for ";

        }

        cout << "student " << student << ": ";
        cin >> score;

        total += score;

    }
    average = total / numTests;

    cout << "The average score for student " << student;

    cout << " is " << average << ".\n\n";

    return 0;
}

```

**Program Output:**

This program averages test scores.

For how many students do you have scores? 2 [Enter]

How many test scores does each student have? 3 [Enter]

Enter score 1 for student 1: 84 [Enter]

Enter score 2 for student 1: 79 [Enter]

Enter score 3 for student 1: 97 [Enter]

The average score for student 1 is 86.7.

Enter score 1 for student 2: 92 [Enter]

Enter score 2 for student 2: 88 [Enter]

Enter score 3 for student 2: 94 [Enter]

The average score for student 2 is 91.3.

**Breaking Out of a Loop**

The break statement causes a loop to terminate early. Sometimes it's necessary to stop a loop before it goes through all its iterations. The break statement, which was used with switch, can also be placed inside a loop. When it is encountered, the loop stops and the program jumps to the statement immediately following the loop. The while loop in the following program segment appears to execute 10 times, but the break statement causes it to stop after the fifth iteration,

**Lab****Tasks:**Task 1

Write a program that uses a loop to display the characters for the ASCII codes 0 to 127. Display 16 characters on each line.

Task 2

Write a program using while loop to find the sum of the following series.  
 $1 + 1/2 + 1/3 + \dots + 1/45$

Task 3

By applying for loop, write a C++ program that prints

A sequence of numbers in ascending order from 1 to 100 (inclusive).

Modify your program in part (a) to make it prints odd numbers within 1 to 100.

Write a C++ Program that receives a positive integer (N) and prints a series of numbers from 1 to N (inclusive) in ascending order.

Write a C++ Program that displays a series of alphabets in descending order from 'Z' to 'A'.

Modify your program in part (d) so that the program will display consonants only, no vowels.

Write a C++ program that receives start value and end value. Then, your program will display a series of numbers from the start value to the end value inclusive.

Task 4

Perform Task 5 from Lab 4 using Do While Loop.

Task 5

(Drawing Patterns with Nested for Loops)

Write a program that uses for statements to print the following patterns separately, one below the other. Use for loops to generate the patterns. All asterisks (\*) should be printed by a single statement of the form `cout << '*';` (this causes the asterisks to print side by side). [Hint: The last two patterns require that each line begin with an appropriate number of blanks. Extra credit: Combine your code from the four separate problems into a single program that prints all four patterns side by side by making clever use of nested for loops.



## Task 6

Draw a pattern of 0's surrounded by \*'s. The pattern should be like.

```

*****
*0*0*0*0*0*
*****
*0*0*0*0*0*
*****

```

## LAB 09: Arrays

### Objective(s):

To Understand about:

Apply Arrays in C++

Use of One Dimensional Array

CLOs: CLO1, CLO4

### INTRODUCTION:

An array works like a variable that can store a group of values, all of the same type. The values are stored together in consecutive memory locations. Here is a definition of an array of integers:

```
int days[6];
```

```
int count;    Enough memory for 1 int
              12314

float price;   Enough memory for 1 float
              56.981

char letter;   Enough memory for 1 char
              A
```

The name of this array is `days`. The number inside the brackets is the array's *size declarator*. It indicates the number of *elements*, or values, the array can hold. The `days` array can store six elements, each one an integer.

An array's size declarator must be a constant integer expression with a value greater than zero. It can be either a literal, as in the previous example, or a named constant, as shown in the following:

```
const int NUM_DAYS = 6;
```

```
int days[NUM_DAYS];
```

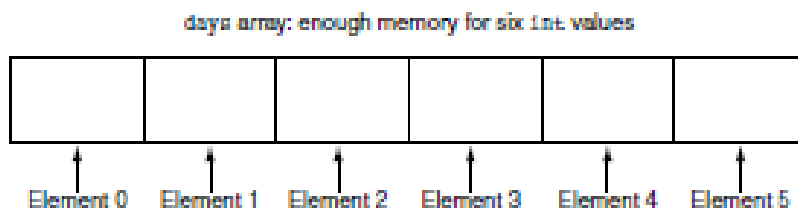
Arrays of any data type can be defined. The following are all valid array definitions:

```
float temperatures[100]; // Array of 100 floats
```

```
char name[41]; // Array of 41 characters
```

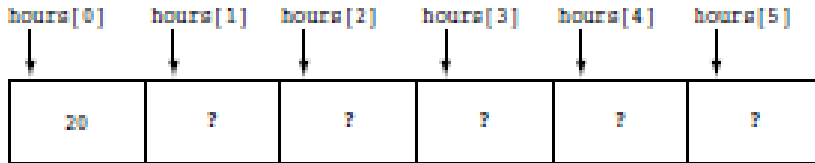
```
long units[50]; // Array of 50 long integers
```

```
double sizes[1200]; // Array of 1200 doubles
```



The following statement stores the integer 30 in `hours[3]`.

```
hours[3] = 30;
```



Example:

```
// This program asks for the number of hours worked
// by six employees. It stores the values in an array.
#include <iostream>
using namespace std;

int main()
{
    const int NUM_EMPLOYEES = 6;
    int hours[NUM_EMPLOYEES];

    // Get the hours worked by each employee.
    cout << "Enter the hours worked by "
         << NUM_EMPLOYEES << " employees: ";
    cin >> hours[0];
    cin >> hours[1];
    cin >> hours[2];
    cin >> hours[3];
    cin >> hours[4];
    cin >> hours[5];

    // Display the values in the array.
    cout << "The hours you entered are:";
    cout << " " << hours[0];
    cout << " " << hours[1];
    cout << " " << hours[2];
    cout << " " << hours[3];
    cout << " " << hours[4];
    cout << " " << hours[5] << endl;
    return 0;
}
```

Program Output:

```
Enter the hours worked by 6 employees: 20 12 40 30 30 15 [Enter]
The hours you entered are: 20 12 40 30 30 15
```

Even though the size declarator of an array definition must be a constant or a literal, subscript numbers can be stored in variables. This makes it possible to use a loop to “cycle through” an entire array, performing the same operation on each element. For example, look at the following code:

```
const int ARRAY_SIZE = 5;
int numbers[ARRAY_SIZE];
for (int count = 0; count < ARRAY_SIZE; count++)
    numbers[count] = 99;
```

This code first defines a constant int named ARRAY\_SIZE and initializes it with the value 5. Then it defines an int array named numbers, using ARRAY\_SIZE as the size declarator. As a result, the numbers array will have five elements. The for loop uses a counter variable

named count. This loop will iterate five times, and during the loop iterations the count variable will take on the values 0 through 4.

Example:

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

const int cARY_SIZE = 5;

int main ()
{
    int sqrAry[cARY_SIZE];
    for (int i = 0; i < cARY_SIZE; i++)
        sqrAry[i] = i * i;

    cout << "Element\tSquare\n";
    cout << "=====\t=====\n";
    for (int i = 0; i < cARY_SIZE; i++)
    {
        cout << setw(5) << i << "\t";
        cout << setw(5) << sqrAry[i] << endl;
    } // for i
    return 0;
} // main
```

Dry Run:

Dry Run			
i	Condition	sqrAry[i]	sqrAry[i]
0	True	0	0
1	True	1	1
2	True	2	4
3	True	3	9
4	True	4	16
5	False		

Program Output:

```
//
Elements      Square
=====
0      0
1      1
2      4
3      9
4     16
```

Output



**Lab Tasks:**

- 1) Array 10 elements
  1. Input
  2. Output
- 2) Array of 15 elements// user input  
Sum all values in array
- 3) Average of array in task 2
- 4) Array 10 elements; find highest value in the array
- 5) Array 10 elements, find lowest in array
- 6) Create 2 arrays, Array 10 elements each
  1. Input values in arrays
  2. Compare arrays, print equal if these are same otherwise print not equal
- 7) Write a program which performs the following tasks:
  1. Initialize an integer array of 10 elements in function modify( )
  2. In modify( ) multiply each element of array by 3
  3. And display each element of array
- 8) Write a program to search the array element. Enter a value from the keyboard and find out the location of the entered value in the array (Array 10 elements). If the entered number is not found in the array display the message —Number is not found.
- 9) Write a program to input array elements and output is labeled, like this:

```
Enter 5 numbers
a[0]: 11.11
a[1]: 33.33
a[2]: 55.55
a[3]: 77.77
a[4]: 99.99
In reverse order, they are:
a[4] = 99.99
a[3] = 77.77
a[2] = 55.55
a[1] = 33.33
a[0] = 11.11
```

## LAB 10: Arrays

### Objective(s):

To Understand about:

Apply Arrays in C++

Use of Two and Three Dimensional Array

**CLOs:** CLO1, CLO4

### 2D Array:

An array is useful for storing and working with a set of data. Sometimes, though, it's necessary to work with multiple sets of data. For example, in a grade-averaging program a teacher might record all of one student's test scores in an array of doubles. If the teacher has 30 students, that means she'll need 30 arrays of doubles to record the scores for the entire class. Instead of defining 30 individual arrays, however, it would be better to define a two-dimensional array.

The arrays that you have studied so far are one-dimensional arrays. They are called *one dimensional* because they can only hold one set of data. Two-dimensional arrays, which are sometimes called *2D arrays*, can hold multiple sets of data. It's best to think of a two dimensional array as having rows and columns of elements, as shown in Figure. This figure shows an array of test scores, having three rows and four columns.

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>scores[0] [0]</code>	<code>scores[0] [1]</code>	<code>scores[0] [2]</code>	<code>scores[0] [3]</code>
Row 1	<code>scores[1] [0]</code>	<code>scores[1] [1]</code>	<code>scores[1] [2]</code>	<code>scores[1] [3]</code>
Row 2	<code>scores[2] [0]</code>	<code>scores[2] [1]</code>	<code>scores[2] [2]</code>	<code>scores[2] [3]</code>

To define a two-dimensional array, two size declarators are required. The first one is for the number of rows and the second one is for the number of columns. Here is an example definition of a two-dimensional array with three rows and four columns:

The first size declarator specifies the number of rows, and the second size declarator specifies the number of columns. Notice that each number is enclosed in its own set of brackets. When processing the data in a two-dimensional array, each element has two subscripts: one for its row and another for its column. In the scores array defined above, the elements in row 0 are referenced as

`scores[0][0]`

`scores[0][1]`

`scores[0][2]`

`scores[0][3]`

The elements in row 1 are

`scores[1][0]`

`scores[1][1]`

`scores[1][2]`

`scores[1][3]`

And the elements in row 2 are

`scores[2][0]`

```
scores[2][1]
scores[2][2]
scores[2][3]
```

The subscripted references are used in a program just like the references to elements in a single dimensional array, except now you use two subscripts. The first subscript represents the row position, and the second subscript represents the column position. For example, the following statement assigns the value 92.25 to the element at row 2, column 1 of the scores array:

```
scores[2][1] = 92.25;
```

And the following statement displays the element at row 0, column 2:

```
cout << scores[0][2];
```

Example:

```
6 int main()
7 {
8     const int NUM_DIVS = 3;           // Number of divisions
9     const int NUM_QTRS = 4;           // Number of quarters
10    double sales[NUM_DIVS][NUM_QTRS]; // Array with 3 rows and 4 columns.
11    double totalSales = 0;             // To hold the total sales.
12    int div, qtr;                      // Loop counters.
13
14    cout << "This program will calculate the total sales of\n";
15    cout << "all the company's divisions.\n";
16    cout << "Enter the following sales information:\n\n";
17
18    // Nested loops to fill the array with quarterly
19    // sales figures for each division.
20    for (div = 0; div < NUM_DIVS; div++)
21    {
22        for (qtr = 0; qtr < NUM_QTRS; qtr++)
23        {
24            cout << "Division " << (div + 1);
25            cout << ", Quarter " << (qtr + 1) << ": $";
26            cin >> sales[div][qtr];
27        }
28        cout << endl; // Print blank line.
29    }
30
31    // Nested loops used to add all the elements.
32    for (div = 0; div < NUM_DIVS; div++)
33    {
34        for (qtr = 0; qtr < NUM_QTRS; qtr++)
35            totalSales += sales[div][qtr];
36    }
37
38    cout << fixed << showpoint << setprecision(2);
39    cout << "The total sales for the company are: $";
40    cout << totalSales << endl;
41    return 0;
42 }
```

```

Program Output with Example Input Shown in Bold
This program will calculate the total sales of
all the company's divisions.
Enter the following sales data:

Division 1, Quarter 1: $31569.45 [Enter]
Division 1, Quarter 2: $29654.23 [Enter]
Division 1, Quarter 3: $32982.54 [Enter]
Division 1, Quarter 4: $39651.21 [Enter]

Division 2, Quarter 1: $56321.02 [Enter]
Division 2, Quarter 2: $54128.63 [Enter]
Division 2, Quarter 3: $41235.85 [Enter]
Division 2, Quarter 4: $54652.33 [Enter]

```

### 3D Array:

Three dimensional (3D) array contains three for loops in programming. So, to initialize and print three dimensional array, you have to use three for loops. Third for loop (the innermost loop) forms 1D array, Second for loop forms 2D array and the third for loop (the outermost loop) forms 3D array, as shown here in the following program.

Example:

```

#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int arr[3][4][2] = {
        {
            {2, 4},
            {7, 8},
            {3, 4},
            {5, 6}
        },
        {
            {7, 6},
            {3, 4},
            {5, 3},
            {2, 3}
        },
        {
            {8, 9},
            {7, 2},
            {3, 4},
            {5, 1}
        }
    };
    cout<<"arr[0][0][0] = "<<arr[0][0][0]<<"\n";
    cout<<"arr[0][2][1] = "<<arr[0][2][1]<<"\n";
    cout<<"arr[2][3][1] = "<<arr[2][3][1]<<"\n";
    getch();
}

```

Output:

```
arr[0][0][0] = 2  
arr[0][2][1] = 4  
arr[2][3][1] = 1
```

**Lab Tasks:**

Array: ROWS=4, COLS=5

1. Summing all values of 2D Array
2. Summing the rows of 2D array
3. Summing the columns of 2D array
4. Get highest in 2D array
5. Get lowest in 2D array
6. get highest in specified row
7. get lowest in specified row
8. Add Two Matrices(2\*2) using 2D Arrays
9. Subtract Two Matrices(2\*2) using 2D Arrays

## LAB 11: Functions

### **Objective(s):**

To Understand about:

Apply user defined functions in C++

1. Function declaration, calling and definition.
2. Functions with Return Value

**CLOs:** CLO1, CLO4

### **INTRODUCTION:**

The best way to develop and maintain a large program is to construct it from small, simple pieces, or components. This technique is called divide and conquer. Functions allow you to modularize a program by separating its tasks into self-contained units. Functions you write are referred to as user-defined functions or programmer-defined functions. The statements in function bodies are written only once, are reused from perhaps several locations in a program and are hidden from other functions.

### **Explanation**

A **-Function Prototype** tells the compiler about a function's name, function's return type and function parameters.

A **-Function Call** calls a function, program control is transferred to the called function (Function definition). A called function performs defined tasks and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

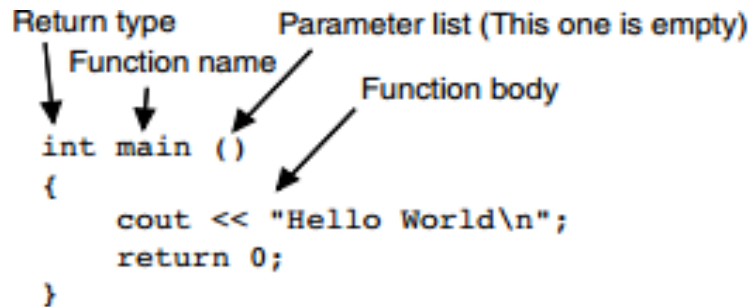
A **-Function Definition** consists of a function header and a function body. Here are all the parts of a function Definition:

**Return Type:** A function may return a value. The return\_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword void.

**Function Name:** This is the actual name of the function.

**Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

**Function Body:** The function body contains a collection of statements that define what statements to execute.



The line in the definition that reads `int main()` is called the *function header*.

### Example:

```
// This program has two functions: main and displayMessage  
#include <iostream>  
using namespace std;  
  
//*****  
// Definition of function displayMessage *  
// This function displays a greeting. *  
//*****  
  
void displayMessage()  
{  
    cout << "Hello from the function displayMessage.\n";  
}  
  
//*****  
// Function main *  
//*****  
  
int main()  
{  
    cout << "Hello from main.\n";  
    displayMessage();  
    cout << "Back in function main again.\n";  
    return 0;  
}
```

### Program Output:

```
Program Output  
Hello from main.  
Hello from the function displayMessage.  
Back in function main again.
```

### Function Prototypes:

A function prototype eliminates the need to place a function definition before all calls to the function.

### Example:

```

// This program has three functions: main, first, and second.
#include <iostream>
using namespace std;

// Function Prototypes
void first();
void second();

int main()
{
    cout << "I am starting in function main.\n";
    first();    // Call function first
    second();   // Call function second
    cout << "Back in function main again.\n";
    return 0;
}

//*****
// Definition of function first.      *
// This function displays a message. *
//*****

void first()
{
    cout << "I am now inside the function first.\n";
}

//*****
// Definition of function second.     *
// This function displays a message. *
//*****

void second()
{
    cout << "I am now inside the function second.\n";
}

```

### Program Output:

#### **Program Output**

(The program's output is the same as the output of Program 6-3.)

### Sending Data into a Function:

When a function is called, the program may send values into the function



```
// This program demonstrates a function with a parameter.
#include <iostream>
using namespace std;

// Function Prototype
void displayValue(int);

int main()
{
    cout << "I am passing 5 to displayValue.\n";
    displayValue(5); // Call displayValue with argument 5
    cout << "Now I am back in main.\n";
    return 0;
}
```

```

//*****
// Definition of function displayValue.          *
// It uses an integer parameter whose value is displayed. *
//*****

void displayValue(int num)
{
    cout << "The value is " << num << endl;
}

```

### Program Output:

```
Program Output
I am passing 5 to displayValue.
The value is 5
Now I am back in main.
```

### Value Returning Functions:

The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function:

#### Example:

```
int myFunction(int x) {

    return 5 + x;

}
```

```
int main() {  
    cout << myFunction(3);  
    return 0;  
}
```

// Outputs 8 (5 + 3)

```
int myFunction(int x, int y) {  
    return x + y;  
}
```

**Example:**

```
int main() {  
    cout << myFunction(5, 3);  
    return 0;  
}
```

// Outputs 8 (5 + 3)

**Local and Global Variables:**

Global variables are declared outside any function, and they can be accessed (used) on any function in the program. Local variables are declared inside a function, and can be used only inside that function. It is possible to have local variables with the same name in different functions. Even the name is the same, they are not the same. It's like two people with the same name. Even the name is the same, the persons are not.

The scope of a variable refers to where a variable is visible or accessible. If a person asks what the scope of a variable is, she's asking whether it is local or global.

// What is a global variable? A local variable? Scope?

// Global variable

```
float a = 1;
```

```
void my_test() {  
    // Local variable called b.  
    // This variable can't be accessed in other functions  
    float b = 77;  
    println(a);  
    println(b);  
}
```

```
void setup() {  
    // Local variable called b.  
    // This variable can't be accessed in other functions.  
    float b = 2;  
    println(a);  
    println(b);  
    my_test();  
    println(b);  
}
```

// You can have local variables with the same name in different functions.

// It's like having two persons with the same name: they are different persons!

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters	Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Accessed by	Local variables can be accessed with the help of statements, inside a function in which they are declared.	You can access global variables by any statement in the program.
Memory storage	It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

#### **Advantages of using Global variables**

You can access the global variable from all the functions or modules in a program

You only require declaring global variable single time outside the modules.

It is ideally used for storing "constants" as it helps you keep the consistency.

A Global variable is useful when multiple functions are accessing the same data.

#### **Advantages of using Local Variables**

The use of local variables offer a guarantee that the values of variables will remain intact while the task is running

If several tasks change a single variable that is running simultaneously, then the result may be unpredictable. But declaring it as local variable solves this issue as each task will create its own instance of the local variable.

You can give local variables the same name in different functions because they are only recognized by the function they are declared in.

Local variables are deleted as soon as any function is over and release the memory space which it occupies.

#### **Disadvantages of using Global Variables**

Too many variables declared as global, and then they remain in the memory till program execution is completed. This can cause of Out of Memory issue.

Data can be modified by any function. Any statement written in the program can change the value of the global variable. This may give unpredictable results in multi-tasking environments.

#### **Static Variables:**

Static variables in a Function: When a variable is declared as static, space for it gets allocated for the lifetime of the program. Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call

gets carried through the next function call. This is useful for implementing coroutines in C/C++ or any other application where previous state of function needs to be stored.

```
// C++ program to demonstrate
```

```
// the use of static Static
```

```
// variables in a Function
```

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
void demo()
```

```
{
```

```
    // static variable
```

```
    static int count = 0;
```

```
    cout << count << " ";
```

```
    // value is updated and
```

```
    // will be carried to next
```

```
    // function calls
```

```
    count++;
```

```
}
```

```
int main()
```

```
{
```

```
    for (int i=0; i<5; i++)
```

```
        demo();
```

```
    return 0;
```

```
}
```

Output:

0 1 2 3 4

## Lab Tasks:

### Task 1

Write a C++ Program that contains four user defined function(s): addition(), subtraction(), division(), multiplication(). Develop a calculator as follows:

#### In main() function:

- A menu with choices addition, subtraction, division and multiplication must be displayed.
- Get two numbers and a choice from user
- Call the respective functions with user given number as parameter using switch statement
- Print the result from addition (), subtraction (), division (), multiplication().

#### In user defined functions:

- Plus and minus function get two integer values and return integer.
- Multiply and Divide functions get two integer values and return float.

### Task2:

Write a C++ Program that contains one user defined function cal\_grades().

#### In main() function:

- Prompt user to enter obtained (0 - 100) marks for one subject.
- Call cal\_grades(marks\_subject).
- Print the corresponding Grade with respect to Marks.

#### In user defined function:

- Perform conditioning with else if statement return char value.
- Function must return value.

### Task3:

Write a program that asks the user to enter an item's wholesale cost and its markup percentage. It should then display the item's retail price.

For example:

- If an item's wholesale cost is 5.00 and its markup percentage is 100%, then the item's retail price is 10.00.

If an item's wholesale cost is 5.00 and its markup percentage is 50%, then the item's retail price is 7.50.

The program should have a function named calculateRetail that receives the wholesale cost and the markup percentage as arguments and returns the retail price of the item.

#### You can use this formula:

$$\text{retailPrice} = (\text{wholesaleCost} * \text{markupPercent}) + \text{wholesaleCost}$$

**Input Validation:** Do not accept negative values for either the wholesale cost of the item or the markup percentage.

## LAB 12: Functions

### **Objective(s):**

To Understand about:

Apply user defined functions in C++

1. Default arguments.
2. Using reference variable as parameters
3. Overloading Functions

**CLOs:** CLO1, CLO4

### **Default Arguments:**

Default arguments are passed to parameters automatically if no argument is provided in the function call.

It's possible to assign default arguments to function parameters. A default argument is passed to the parameter when the actual argument is left out of the function call. The default arguments are usually listed in the function prototype. Here is an example:

```
void showArea(double = 20.0, double = 10.0);
```

Default arguments are literal values or constants with an = operator in front of them, appearing after the data types listed in a function prototype. Since parameter names are optional in function prototypes, the example prototype could also be declared as

```
void showArea(double length = 20.0, double width = 10.0);
```

It's possible to assign default arguments to function parameters. A default argument is passed to the parameter when the actual argument is left out of the function call. The default arguments are usually listed in the function prototype. Here is an example:

```
void showArea(double = 20.0, double = 10.0);
```

Default arguments are literal values or constants with an = operator in front of them, appearing after the data types listed in a function prototype. Since parameter names are optional in function prototypes, the example prototype could also be declared as

```
void showArea(double length = 20.0, double width = 10.0);
```

In both example prototypes, the function showArea has two double parameters. The first is assigned the default argument 20.0 and the second is assigned the default argument 10.0. Here is the definition of the function:

```
void showArea(double length, double width)
{
    double area = length * width;
    cout << "The area is " << area << endl;
}
```

The default argument for length is 20.0 and the default argument for width is 10.0. Because both parameters have default arguments, they may optionally be omitted in the

function call, as shown here:

```
showArea();
```

In this function call, both default arguments will be passed to the parameters. The parameter

length will take the value 20.0 and width will take the value 10.0. The output of the function will be

The area is 200

The default arguments are only used when the actual arguments are omitted from the function

call. In the call below, the first argument is specified, but the second is omitted:

```
showArea(12.0);
```

The value 12.0 will be passed to length, while the default value 10.0 will be passed to width. The output of the function will be

The area is 120

Of course, all the default arguments may be overridden. In the function call below, arguments are supplied for both parameters:

```
showArea(12.0, 5.5);
```

The output of the function call above will be

The area is 66

### **Example: Default Argument**

```
#include <iostream>
```

```
using namespace std;
```

```
// defining the default arguments
```

```
void display(char = '*', int = 3);
```

```
int main() {
```

```
    int count = 5;
```

```
    cout << "No argument passed: ";
```

```
    // *, 3 will be parameters
```

```
    display();
```

```
    cout << "First argument passed: ";
```

```
    // #, 3 will be parameters
```

```
    display('#');
```

```
    cout << "Both arguments passed: ";
```

```
    // $, 5 will be parameters
```

```
    display('$', count);
```

```
    return 0;
```

```
}
```

```
void display(char c, int count) {
```



```

for(int i = 1; i <= count; ++i)
{
    cout << c;
}

```

### Output:

No argument passed: \*\*\*

First argument passed: ###

Both arguments passed: \$\$\$\$

### Things to Remember:

Once we provide a default value for a parameter, all subsequent parameters must also have default values. For example,

// Invalid

```
void add(int a, int b = 3, int c, int d);
```

// Invalid

```
void add(int a, int b = 3, int c, int d = 4);
```

// Valid

```
void add(int a, int c, int b = 3, int d = 4);
```

If we are defining the default arguments in the function definition instead of the function prototype, then the function must be defined before the function call.

// Invalid code

```

int main() {
    // function call
    display();
}

```

```

void display(char c = '*', int count = 5) {
    // code
}

```

In C++ we can pass arguments into a function in different ways. These different ways are

- Call by Value
- Call by Reference
- Call by Address

Sometimes the call by address is referred to as call by reference, but they are different in C++. In call by address, we use pointer variables to send the exact memory address, but in call by reference we pass the reference variable (alias of that variable). This feature is not present in C, there we have to pass the pointer to get that effect. In this section we will see what are the advantages of call by reference over call by value, and where to use them

**Call by Value:**

In call by value, the actual value that is passed as argument is not changed after performing some operation on it. When call by value is used, it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same.

**Example:**

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

```
void my_function(int x) {
    x = 50;
    cout << "Value of x from my_function: " << x << endl;
}
```

```
main() {
    int x = 10;
    my_function(x);
    cout << "Value of x from main function: " << x;
}
```

**Output:**

```
Value of x from my_function: 50
Value of x from main function: 10
```

**Call by Reference**

In call by reference the actual value that is passed as argument is changed after performing some operation on it. When call by reference is used, it creates a copy of the reference of that variable into the stack section in memory. It uses a reference to get the value. So, when the value is changed using the reference it changes the value of the actual variable.

**Example:**

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

```
void my_function(int &x) {
    x = 50;
    cout << "Value of x from my_function: " << x << endl;
}
```

```
main() {
    int x = 10;
    my_function(x);
    cout << "Value of x from main function: " << x;
}
```

**Output:**

Value of x from my\_function: 50

Value of x from main function: 50

**Where to use Call by reference?**

The call by reference is mainly used when we want to change the value of the passed argument into the invoker function.

One function can return only one value. When we need more than one value from a function, we can pass them as an output argument in this manner.

**Function Overloading:**

With function overloading, multiple functions can have the same name with different parameters:

**Example:**

```
int myFunction(int x)
```

```
float myFunction(float x)
```

```
double myFunction(double x, double y)
```

Consider the following example, which have two functions that add numbers of different type:

**Example:**

```
int plusFuncInt(int x, int y) {  
    return x + y;  
}
```

```
double plusFuncDouble(double x, double y) {  
    return x + y;  
}
```

```
int main() {  
    int myNum1 = plusFuncInt(8, 5);  
    double myNum2 = plusFuncDouble(4.3, 6.26);  
    cout << "Int: " << myNum1 << "\n";  
    cout << "Double: " << myNum2;  
    return 0;  
}
```

Instead of defining two functions that should do the same thing, it is better to overload one. In the example below, we overload the plusFunc function to work for both int and double:

**Example**

```
int plusFunc(int x, int y) {  
    return x + y;  
}
```

```
double plusFunc(double x, double y) {  
    return x + y;  
}
```

```
int main() {  
    int myNum1 = plusFunc(8, 5);  
    double myNum2 = plusFunc(4.3, 6.26);  
    cout << "Int: " << myNum1 << "\n";  
    cout << "Double: " << myNum2;  
    return 0;  
}
```

Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

## Lab Tasks:

### Task 1:

Write a program that computes and displays the charges for a patient's hospital stay. First, the program should ask if the patient was admitted as an in-patient or an outpatient. If the patient was an in-patient, the following data should be entered:

- The number of days spent in the hospital
- The daily rate
- Hospital medication charges
- Charges for hospital services (lab tests, etc.)

The program should ask for the following data if the patient was an out-patient:

- Charges for hospital services (lab tests, etc.)
- Hospital medication charges

The program should use two overloaded functions to calculate the total charges. One of the functions should accept arguments for the in-patient data, while the other function accepts arguments for out-patient information. Both functions should return the total charges.

**Input Validation:** Do not accept negative numbers for any data.

### Task2: Population

In a population, the birth rate is the percentage increase of the population due to births, and the death rate is the percentage decrease of the population due to deaths. Write a program that displays the size of a population for any number of years. The program should ask for the following data:

- The starting size of a population
- The annual birth rate
- The annual death rate
- The number of years to display

Write a function that calculates the size of the population for a year. The formula is  $N = P + BP - DP$

where N is the new population size, P is the previous population size, B is the birth rate, and D is the death rate.

### Task 3: Paint Job Estimator

A painting company has determined that for every 110 square feet of wall space, one gallon of paint and eight hours of labor will be required. The company charges \$25.00 per hour for labor. Write a modular program that allows the user to enter the number of rooms that are to be painted and the price of the paint per gallon. It should also ask for the square feet of wall space in each room. It should then display the following data:

- The number of gallons of paint required
- The hours of labor required
- The cost of the paint
- The labor charges
- The total cost of the paint job

**Input validation:** Do not accept a value less than 1 for the number of rooms. Do not

accept a value less than \$10.00 for the price of paint. Do not accept a negative value for square footage of wall space.

**Task 4:**

Write function which accept an array as argument and increment array's elements by 5, then write a function to show array elements

Note: input array in main function

## LAB 13: Searching and Sorting Arrays

**CLOs:** CLO1, CLO4

### **Searching:**

A search algorithm is a method of locating a specific item in a larger collection of data.

This section discusses two algorithms for searching the contents of an array.

### **The Linear Search:**

The linear search is a very simple algorithm. Sometimes called a sequential search, it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for and stops when either the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm will unsuccessfully search to the end of the array

### **Example:**

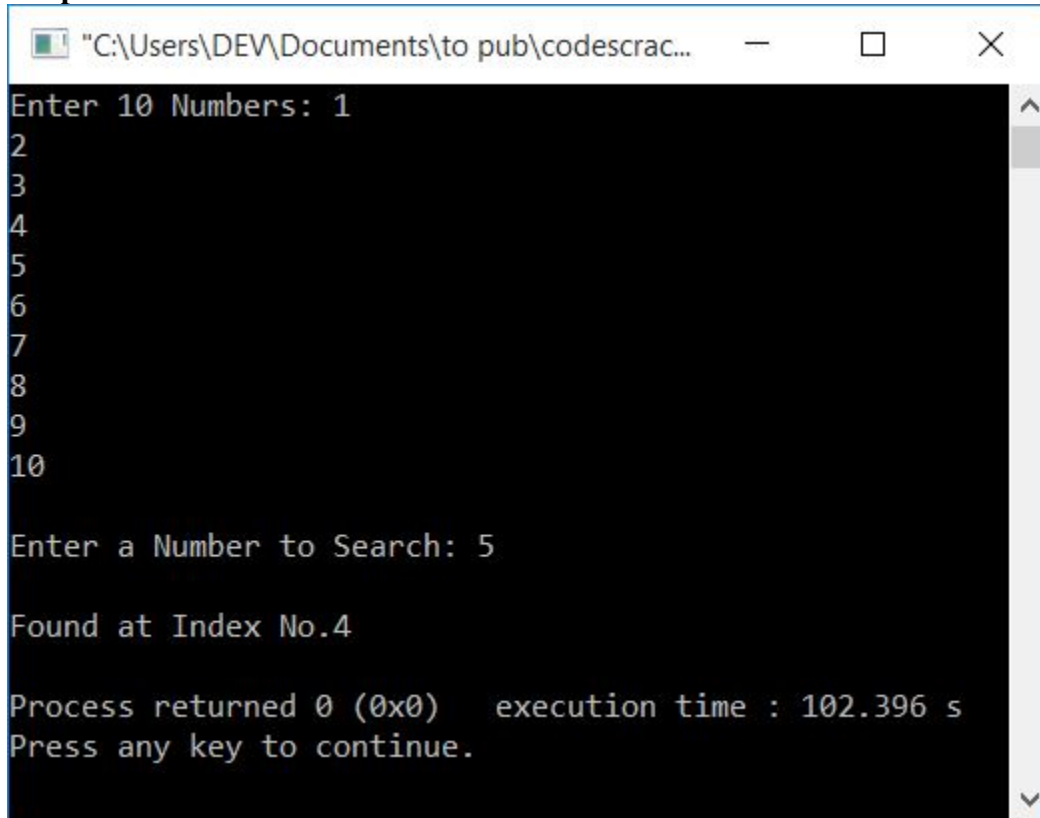
```
#include<iostream>

using namespace std;

int main()
{
    int arr[10], i, num, index;
    cout<<"Enter 10 Numbers: ";
    for(i=0; i<10; i++)
        cin>>arr[i];
    cout<<"\nEnter a Number to Search: ";
    cin>>num;
    for(i=0; i<10; i++)
    {
        if(arr[i]==num)
        {
            index = i;
            break;
        }
    }
    cout<<"\nFound at Index No."<<index;
```

```
cout<<endl;
return 0;
}
```

### Output:



```
"C:\Users\DEV\Documents\to pub\codescrac...
Enter 10 Numbers: 1
2
3
4
5
6
7
8
9
10
Enter a Number to Search: 5
Found at Index No.4
Process returned 0 (0x0) execution time : 102.396 s
Press any key to continue.
```

### Binary Search:

Binary Search: Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

We basically ignore half of the elements just after one comparison.

- Compare x with the middle element.
- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So, we recur for right half.
- Else (x is smaller) recur for the left half.

### Example:

```
#include <bits/stdc++.h>
using namespace std;
```



```

int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? cout << "Element is not present in array"
                  : cout << "Element is present at index " << result;
    return 0;
}

```

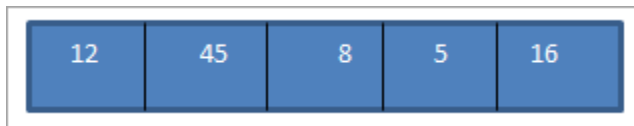
### **Sorting Arrays:**

Sorting is a technique that is implemented to arrange the data in a specific order. Sorting is required to ensure that the data which we use is in a particular order so that we can easily retrieve the required piece of information from the pile of data.

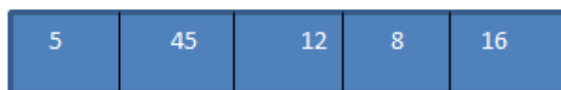
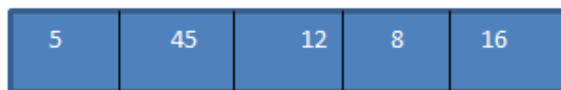
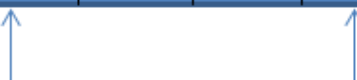
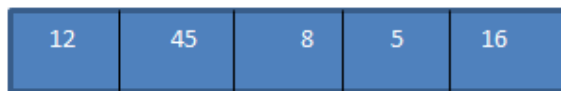
### **Selection Sort:**

It is simple yet easy to implement technique in which we find the smallest element in the list and put it in its proper place. At each pass, the next smallest element is selected and placed in its proper position.

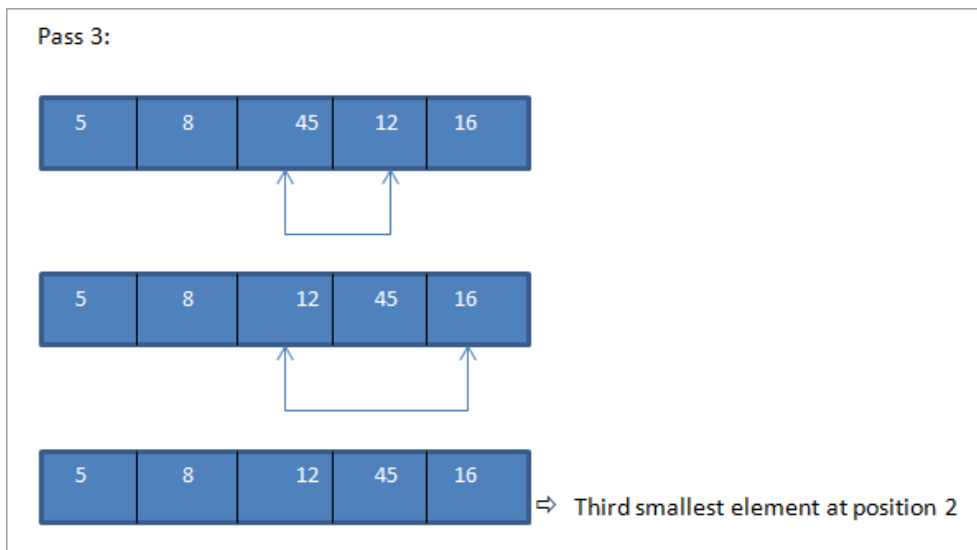
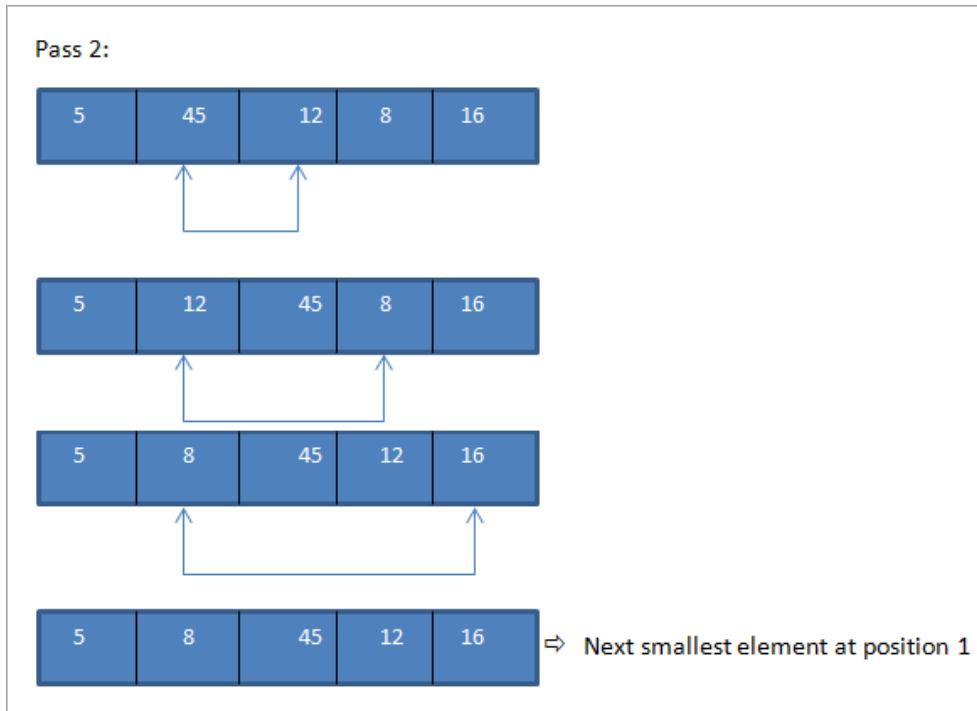
Let us take the same array as in the previous example and perform Selection Sort to sort this array.



**Pass 1:**



⇒ Smallest element at position 0



As shown in the above illustration, for N number of elements we take N-1 passes to completely sort the array. At the end of every pass, the smallest element in the array is placed at its proper position in the sorted array.

**Example:**

```
#include<iostream>
using namespace std;
int findSmallest (int[],int);
int main ()
{
    int myarray[5] = { 12,45,8,15,33};
    int pos,temp;
    cout<<"\n Input list of elements to be Sorted\n";
    for(int i=0;i<5;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    for(int i=0;i<5;i++)
    {
        pos = findSmallest (myarray,i);
        temp = myarray[i];
        myarray[i]=myarray[pos];
        myarray[pos] = temp;
    }
    cout<<"\n Sorted list of elements is\n";
    for(int i=0;i<5;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    return 0;
}
```

```
int findSmallest(int myarray[],int i)
{
    int ele_small,position,j;
    ele_small = myarray[i];
    position = i;
    for(j=i+1;j<5;j++)
    {
        if(myarray[j]<ele_small)
        {
            ele_small = myarray[j];
            position=j;
        }
    }
    return position;
}
```

Output:

Input list of elements to be Sorted

12    45    8    15    33

Sorted list of elements is

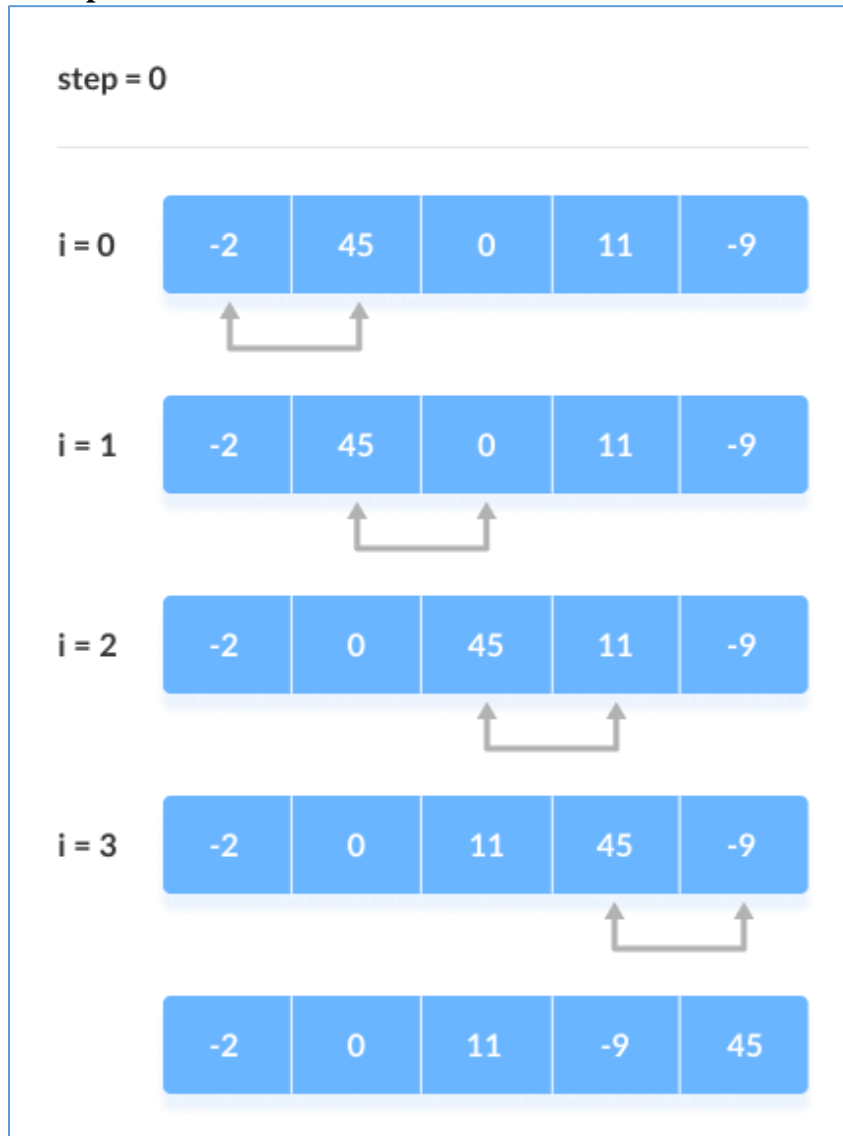
8    12    15    33    45

In selection sort, with every pass, the smallest element in the array is placed in its proper position. Hence at the end of the sorting process, we get a completely sorted array.

**Bubble Sort:**

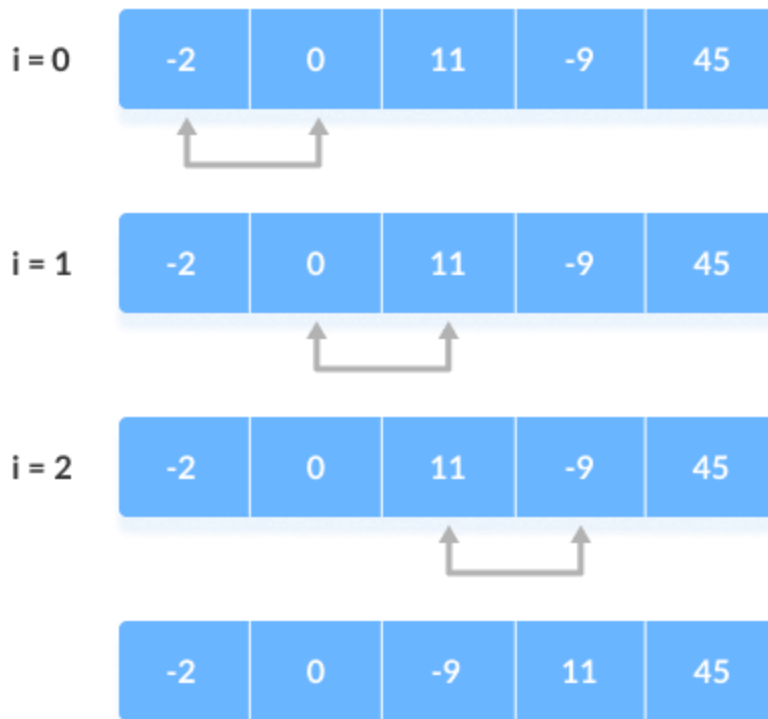
Bubble sort is an algorithm that compares the adjacent elements and swaps their positions if they are not in the intended order. The order can be ascending or descending.

**Example:**



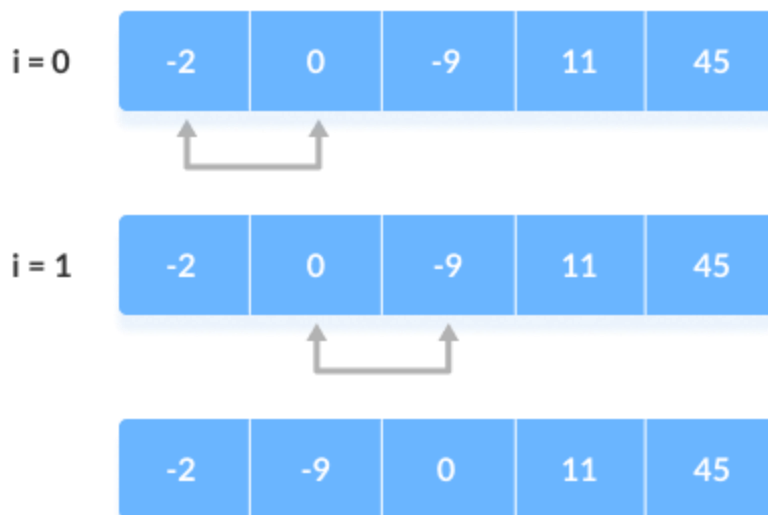
step = 1

---

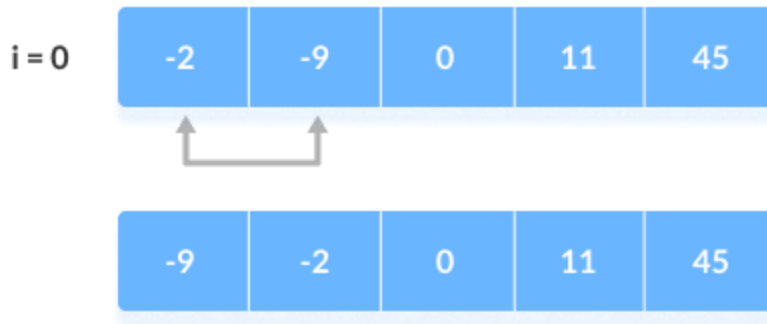


step = 2

---



step = 3



**Example:**

```
#include <iostream>
using namespace std;
void bubbleSort(int array[], int size) {

    // run loops two times: one for walking through the array
    // and the other for comparison
    for (int step = 0; step < size - 1; ++step) {
        for (int i = 0; i < size - step - 1; ++i) {

            // To sort in descending order, change > to < in this line.
            if (array[i] > array[i + 1]) {

                // swap if greater is at the rear position
                int temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
            }
        }
    }
}

// function to print the array
void printArray(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << " " << array[i];
    }
    cout << "\n";
}
```

```
int main() {  
    int data[] = {-2, 45, 0, 11, -9};  
    int size = sizeof(data) / sizeof(data[0]);  
    bubbleSort(data, size);  
    cout << "Sorted Array in Ascending Order:\n";  
    printArray(data, size);  
}
```



## **Lab Tasks:**

### **Task1: Charge Account Validation**

Write a program that lets the user enter a charge account number. The program should determine if the number is valid by checking for it in the following list:

5658845 4520125 7895122 8777541 8451277 1302850

8080152 4562555 5552012 5050552 7825877 1250255

1005231 6545231 3852085 7576651 7881200 4581002

The list of numbers above should be initialized in a single-dimensional array. A simple linear search should be used to locate the number entered by the user. If the user enters a number that is in the array, the program should display a message saying that the number is valid. If the user enters a number that is not in the array, the program should display a message indicating that the number is invalid.

### **Task 2:**

A lottery ticket buyer purchases 10 tickets a week, always playing the same 10 5-digit “lucky” combinations. Write a program that initializes an array with these numbers and then lets the player enter this week’s winning 5-digit number.

The program should perform a linear search through the list of the player’s numbers and report whether or not one of the tickets is a winner this week. Here are the numbers:

13579, 26791, 26792, 33445, 55555

62483 77777 79422 85647 93121

### **Task 3:**

Modify the program you wrote for Task 2 (Lottery Winners) so it performs a binary search instead of a linear search.

### **Task 4:**

Modify the program you wrote for Task 1 (Charge Account Validation) so it performs a binary search to locate valid account numbers. Use the selection sort algorithm to sort the array before the binary search is performed.

### **Task 5:**

Write a program that uses two identical arrays of at least 20 integers. It should call a function that uses the bubble sort algorithm to sort one of the arrays in ascending order. The function should keep a count of the number of exchanges it makes. The program then should call a function that uses the selection sort algorithm to sort the other array. It should also keep count of the number of exchanges it makes. Display these values on the screen.

## LAB 14: File Handling

### **Objective(s):**

To Understand about:

1. Able to read from a file
2. Able to write in a file

**CLOs:** CLO1, CLO4

### **Introduction**

The programs you have written so far require you to re-enter data each time the program runs. This is because the data stored in RAM disappears once the program stops running or the computer is shut down. If a program is to retain data between the times it runs, it must have a way of saving it. Data is saved in a file, which is usually stored on a computer's disk. Once the data is saved in a file, it will remain there after the program stops running. The data can then be retrieved and used at a later time.

There are always three steps that must be taken when a file is used by a program:

1. The file must be *opened*. If the file does not yet exist, opening it means creating it.
2. Data is then saved to the file, read from the file, or both.
3. When the program is finished using the file, the file must be *closed*.

### **Setting up a Program for File Input/Output**

C++ provides the following classes to perform output and input of characters to/from files:

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files.

File Stream	
Data Type	Description
ofstream	Output file stream. This data type can be used to create files and write data to them. With the ofstream data type, data may only be copied from variables to the file, but not vice versa.
ifstream	Input file stream. This data type can be used to open existing files and read data from them into memory. With the ifstream data type, data may only be copied from the file into variables, not but vice versa.
fstream	File stream. This data type can be used to create files, write data to them, and read data from them. With the fstream data type, data may be copied from variables into a file, or from a file into variables.

These classes are derived directly or indirectly from the classes istream, and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files. Let's see an example:

<pre>// basic file operations #include &lt;iostream&gt; #include &lt;fstream&gt; using namespace std;  int main () {     ofstream myfile;     myfile.open ("example.txt");     myfile &lt;&lt; "Writing this to a file.\n";     myfile.close();     return 0; }</pre>	<pre>[file example.txt] Writing this to a file.</pre>
---	---

This code creates a file called example.txt and inserts a sentence into it in the same way we are used to do with cout, but using the file stream myfile instead.

But let's go step by step:

### Open a file

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file. An open file is represented within a program by a stream object (an instantiation of one of these classes, in the previous example this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open():

open (filename, mode);

Where filename is a null-terminated character sequence of type const char \* (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
ios::trunc	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

All these flags can be combined using the bitwise operator OR (`|`). For example, if we want to open the file `example.bin` in binary mode to add data we could do it by the following call to member function `open()`:

```
ofstream myfile;
```

```
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each one of the `open()` member functions of the classes `ofstream`, `ifstream` and `fstream` has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
<code>ofstream</code>	<code>ios::out</code>
<code>ifstream</code>	<code>ios::in</code>
<code>fstream</code>	<code>ios::in   ios::out</code>

For `ifstream` and `ofstream` classes, `ios::in` and `ios::out` are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the `open()` member function.

The default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

File streams opened in binary mode perform input and output operations independently of any format considerations. Non-binary files are known as text files, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

Since the first task that is performed on a file stream object is generally to open a file, these three classes include a constructor that automatically calls the `open()` member function and has the exact same parameters as this member. Therefore, we could also have declared the previous `myfile` object and conducted the same opening operation in our previous example by writing:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.

To check if a file stream was successful opening a file, you can do it by calling to member `is_open()` with no arguments. This member function returns a `bool` value of `true` in the case that indeed the stream object is associated with an open file, or `false` otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

### Closing a file

When we are finished with our input and output operations on a file we shall close it so that its resources become available again. In order to do that we have to call the stream's member function `close()`. This member function takes no parameters, and what it does is to flush the associated buffers and close the file:

```
myfile.close();
```

Once this member function is called, the stream object can be used to open another file, and the file is available again to be opened by other processes.

## Text files

Text file streams are those where we do not include the `ios::binary` flag in their opening mode. These files are designed to store text and thus all values that we input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Data output operations on text files are performed in the same way we operated with `cout`

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ofstream myfile ("example.txt");
    if (myfile.is_open())
    {
        myfile << "This is a line.\n";
        myfile << "This is another line.\n";
        myfile.close();
    }
    else cout << "Unable to open file";
    return 0;
}
```

```
[file example.txt]
This is a line.
This is another line.
```

Data input from a file can also be performed in the same way that we did with `cin`:

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open())
    {
        while ( getline (myfile,line) )
        {
            cout << line << endl;
        }
        myfile.close();
    }

    else cout << "Unable to open file";

    return 0;
}
```

```
This is a line.
This is another line.
```

This last example reads a text file and prints out its content on the screen. We have created a while loop that reads the file line by line, using `getline`. The value returned by `getline` is a reference to the stream object itself, which when evaluated as a boolean expression (as in this while-loop) is true if the stream is ready for more operations, and false if either the end of the file has been reached or if some other error occurred.

### **Checking state flags**

In addition to `good()`, which checks whether the stream is ready for input/output operations, other member functions exist to check for specific states of a stream (all of them return a `bool` value):

#### **`bad()`**

Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

#### **`fail()`**

Returns true in the same cases as `bad()`, but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

#### **`eof()`**

Returns true if a file open for reading has reached the end.

#### **`good()`**

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

In order to reset the state flags checked by any of these member functions we have just seen we can use the member function `clear()`, which takes no parameters.

### **get and put stream pointers**

All i/o streams objects have, at least, one internal stream pointer:

`ifstream`, like `istream`, has a pointer known as the get pointer that points to the element to be read in the next input operation.

`ofstream`, like `ostream`, has a pointer known as the put pointer that points to the location where the next element has to be written.

Finally, `fstream`, inherits both, the get and the put pointers, from `iostream` (which is itself derived from both `istream` and `ostream`).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

#### **`tellg()` and `tellp()`**

These two member functions have no parameters and return a value of the member type `pos_type`, which is an integer data type representing the current position of the get stream pointer (in the case of `tellg()`) or the put stream pointer (in the case of `tellp()`).

#### **`seekg()` and `seekp()`**

These functions allow us to change the position of the get and put stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

seekg ( position );

seekp ( position );

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos\_type, which is an integer value.

The other prototype for these functions is:

seekg ( offset, direction );

seekp ( offset, direction );

Using this prototype, the position of the get or put pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off\_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from, and that can take any of the following values:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position of the stream pointer
ios::end	offset counted from the end of the stream

The following example uses the member functions we have just seen to obtain the size of a file:

```
/ obtaining file size
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    long begin,end;
    ifstream myfile ("example.txt");
    begin = myfile.tellg();
    myfile.seekg (0, ios::end);
    end = myfile.tellg();
    myfile.close();
    cout << "size is: " << (end-begin) << "
bytes.\n";
    return 0;
}
```

size is: 40 bytes.

### C++ write() function

The write() function is used to write object or record (sequence of bytes) to the file. A record may be an array, structure or class.

Syntax of write() function

```
fstream fout;  
fout.write( (char *) &obj, sizeof(obj) );
```

The write() function takes two arguments.

&obj : Initial byte of an object stored in memory.

sizeof(obj) : size of object represents the total number of bytes to be written from initial byte.

### **C++ read() function**

The read() function is used to read object (sequence of bytes) to the file.

#### **Syntax of read() function**

```
fstream fin;  
fin.read( (char *) &obj, sizeof(obj) );
```

The read() function takes two arguments.

&obj : Initial byte of an object stored in file.

sizeof(obj) : size of object represents the total number of bytes to be read from initial byte.

The read() function returns NULL if no data read.



**Lab Tasks:****Task 1:**

Write a program, which asks the user to enter name of 10 employees, store the names in file. Display total number of words in the file.

**Task 2:**

Write a program that calculates the balance of a savings account at the end of a period of time. It should ask the user for the annual interest rate, the starting balance, and the number of months that have passed since the account was established. A loop should then iterate once for every month, performing the following:

- Ask the user for the amount deposited into the account during the month. (Do not accept negative numbers.) This amount should be added to the balance.
- Ask the user for the amount withdrawn from the account during the month. (Do not accept negative numbers.) This amount should be subtracted from the balance.
- Calculate the monthly interest. The monthly interest rate is the annual interest rate divided by twelve. Multiply the monthly interest rate by the balance, and add the result to the balance.

After the last iteration, the program should display the ending balance, the total amount of deposits, the total amount of withdrawals, and the total interest earned.

**NOTE:** If a negative balance is calculated at any point, a message should be displayed indicating the account has been closed and the loop should terminate.

## LAB 15: Pointers

### **Objective(s):**

To Understand about:

1. Able to use Pointers
2. Able to use pointers in functions

**CLOs:** CLO1, CLO4

### **Introduction**

Pointers are powerful features of C++ that differentiates it from other programming languages like Java and Python.

Pointers are used in C++ program to access the memory and manipulate the address.

### **Address in C++**

To understand pointers, you should first know how data is stored on the computer. Each variable you create in your program is assigned a location in the computer's memory. The value the variable stores is actually stored in the location assigned. To know where the data is stored, C++ has an & operator. The & (reference) operator gives you the address occupied by a variable. If var is a variable then, &var gives the address of that variable.

### **Example : Address in C++**

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

int main()
{
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;
    cout << &var1 << endl;
    cout << &var2 << endl;
    cout << &var3 << endl;
}
```

### **Output**

```
0x7fff5fbff8ac
0x7fff5fbff8a8
0x7fff5fbff8a4
```

Note: You may not get the same result on your system.

The 0x in the beginning represents the address is in hexadecimal form.

Notice that first address differs from second by 4-bytes and second address differs from third by 4-bytes.

This is because the size of integer (variable of type int) is 4 bytes in 64-bit system.

## Pointers Variables

C++ gives you the power to manipulate the data in the computer's memory directly. You can assign and de-assign any space in the memory as you wish. This is done using Pointer variables.

Pointers variables are variables that points to a specific address in the memory pointed by another variable.

### How to declare a pointer?

```
int *p;  
    OR,  
int* p;
```

The statement above defines a pointer variable p. It holds the memory address

The asterisk is a dereference operator which means pointer to.

Here, pointer p is a pointer to int, i.e., it is pointing to an integer value in the memory address.

### Reference operator (&) and Deference operator (\*)

Reference operator (&) as discussed above gives the address of a variable.

To get the value stored in the memory address, we use the dereference operator (\*).

For example: If a number variable is stored in the memory address 0x123, and it contains a value 5.

The reference (&) operator gives the value 0x123, while the dereference (\*) operator gives the value 5.

Note: The (\*) sign used in the declaration of C++ pointer is not the dereference pointer. It is just a similar notation that creates a pointer.

### Example: C++ Pointers

C++ Program to demonstrate the working of pointer.

```
#include <iostream>  
using namespace std;  
int main() {  
    int *pc, c;  
  
    c = 5;  
    cout << "Address of c (&c): " << &c << endl;  
    cout << "Value of c (c): " << c << endl << endl;  
  
    pc = &c; // Pointer pc holds the memory address of variable c  
    cout << "Address that pointer pc holds (pc): " << pc << endl;  
    cout << "Content of the address pointer pc holds (*pc): " << *pc << endl << endl;  
  
    c = 11; // The content inside memory address &c is changed from 5 to 11.  
    cout << "Address pointer pc holds (pc): " << pc << endl;  
    cout << "Content of the address pointer pc holds (*pc): " << *pc << endl << endl;
```

```

*pc = 2;
cout << "Address of c (&c): " << &c << endl;
cout << "Value of c (c): " << c << endl << endl;

return 0;
}

```

## Output

Address of c (&c): 0x7fff5fbff80c

Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff5fbff80c

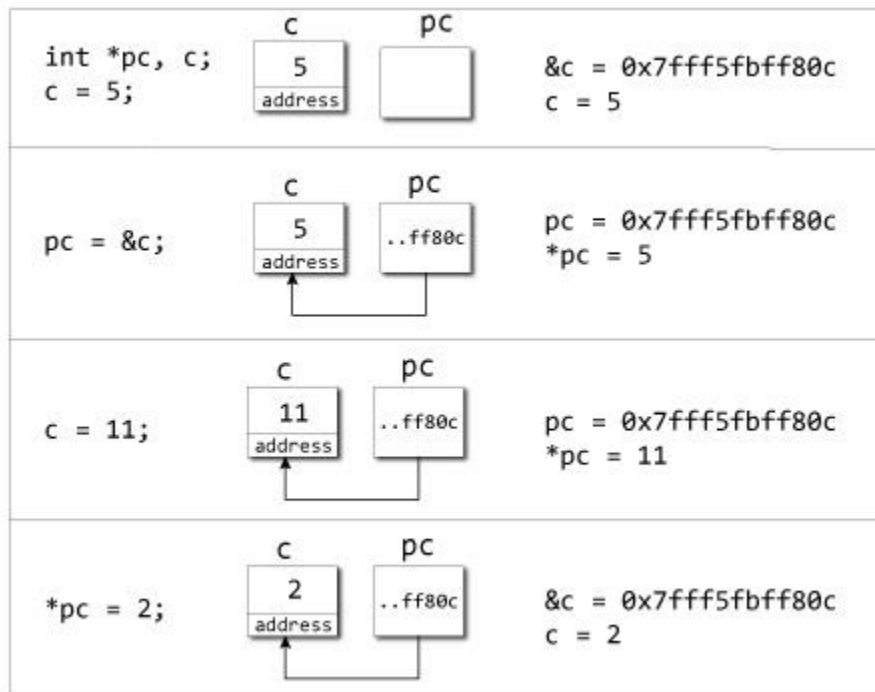
Content of the address pointer pc holds (\*pc): 5

Address pointer pc holds (pc): 0x7fff5fbff80c

Content of the address pointer pc holds (\*pc): 11

Address of c (&c): 0x7fff5fbff80c

Value of c (c): 2



## Explanation of program

- When `c = 5`; the value 5 is stored in the address of variable `c` - 0x7fff5fbff80c.
- When `pc = &c`; the pointer `pc` holds the address of `c` - 0x7fff5fbff80c, and the expression (dereference operator) `*pc` outputs the value stored in that address, 5.
- When `c = 11`; since the address pointer `pc` holds is the same as `c` - 0x7fff5fbff80c, change in the value of `c` is also reflected when the expression `*pc` is executed, which now outputs 11.

- When `*pc = 2;` it changes the content of the address stored by `pc` - `0x7fff5fbff8c`. This is changed from 11 to 2. So, when we print the value of `c`, the value is 2 as well.

### Common mistakes when working with pointers

Suppose, you want pointer `pc` to point to the address of `c`. Then,

```
int c, *pc;
pc=c; /* Wrong! pc is address whereas, c is not an address. */
*pc=&c; /* Wrong! *pc is the value pointed by address whereas, &c is an address. */
pc=&c; /* Correct! pc is an address and, &pc is also an address. */
*pc=c; /* Correct! *pc is the value pointed by address and, c is also a value. */
```

### Passing pointer values in C++

Like any other type, a pointer may be passed as an argument to a function:

```
void fn(int* pnArg)
{
    *pnArg = 10;
}

void parent(void)
{
    int n = 0;
    fn(&n);      // this passes the address of n
                // now the value of n is 10
}
```

In this case, the address of `n` is passed to the function `fn()` rather than the value of `n`. The significance of this difference is apparent when you consider the assignment within `fn()`.

Suppose `n` is located at address `0x100`. Rather than the value 10, the call `fn(&n)` passes the value `0x100`. Within `fn()`, the assignment `*pnArg = 10` stores the value 10 in the `int` variable located at location `0x100`, thereby overwriting the value 0. Upon returning to `parent()`, the value of `n` is 10 because `n` is just another name for `0x100`.

Instead of a regular value or even a reference, a function can return a pointer. You can start to specify this by typing the `*` operator on the left side of the function's name. Here is an example:

```
double * getSalary()

{

}
```

Then, use the body of the function to define it. Before the closing curly bracket of the function, remember to return a pointer to the return value. Here is an example:

```
double * getSalary()

{

    double salary = 26.48;

    double *hourlySalary = &salary;

    return hourlySalary;

}
```

Because a pointer by defining is a reference to the address where a variable resides, when a function is defined as returning a pointer, you can also return a reference to the appropriate type. Here is an example:

```
double * getSalary()

{

    double salary = 26.48;

    return &salary;

}
```

### **Dynamic Memory Allocation:**

Dynamic memory allocation in C/C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack.

One use of dynamically allocated memory is to allocate memory of variable size which is not possible with compiler allocated memory except variable length arrays.

The most important use is flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need and whenever we don't need anymore. There are many cases where this flexibility helps.

C uses malloc() and calloc() function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory. C++ supports these functions and also has two operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

### **new operator**

The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

#### **Syntax to use new operator:**

To allocate memory of any data type, the syntax is:

```
pointer-variable = new data-type;
```

Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class.

Example:

```
// Pointer initialized with NULL
```

```
// Then request memory for the variable
```

```
int *p = NULL;
```

```
p = new int;
```

OR

```
// Combine declaration of pointer
```

```
// and their assignment
```

```
int *p = new int;
```

**Allocate block of memory:** new operator is also used to allocate a block(an array) of memory of type data-type.

```
pointer-variable = new data-type[size];
```

where size(a variable) specifies the number of elements in an array.

Example:

```
int *p = new int[10]
```

Dynamically allocates memory for 10 integers continuously of type int and returns pointer to the first element of the sequence, which is assigned to p(a pointer). p[0] refers to first element, p[1] refers to second element and so on.



## **delete operator**

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

Syntax:

```
// Release memory pointed by pointer-variable
```

```
delete pointer-variable;
```

Here, pointer-variable is the pointer that points to the data object created by new.

Examples:

```
delete p;
```

```
delete q;
```

To free the dynamically allocated array pointed by pointer-variable, use following form of delete:

```
// Release block of memory
```

```
// pointed by pointer-variable
```

```
delete[] pointer-variable;
```

## **Lab Tasks:**

### **Task 1**

Write a program that swaps two values by passing pointers as arguments to function.

### **Task 2**

Write a program that take five input by user in an array and pass it to function. Find the smallest number in an array using pointers.

### **Task 3**

Write a program to find the sum of array and find factorial using sum of array using pointers.

## LAB 16: Structures

### Objective(s):

To Understand about:

1. Apply Structures in C++
2. Use of Structures along with previously used concepts of C++.

**CLO:** CLO1, CLO4

### Introduction

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information's separately. However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2. You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well. This collection of all related information under a single name Person is a structure.

### Declaration of a Structure in C++

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
    char name[50];
    int age;
    float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

When a structure is created, no memory is allocated. The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype. When you define an integer as below:

```
int foo;
```

The int specifies that, variable foo can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

### Defining Variable of a Structure

Once you declare a structure person as above. You can define a structure variable as:

```
Person bill;
```

Here, a structure variable bill is defined which is of type structure Person.

When structure variable is defined, only then the required memory is allocated by the compiler.

Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte.

Hence, 58 bytes of memory is allocated for structure variable bill.

### Accessing members of a Structure

The members of structure variable is accessed using a dot (.) operator.

Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

### Example:

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

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
```

```

    cout << "Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}

```

### Output

```

Enter Full name: Magdalena Dankova
Enter age: 27
Enter salary: 1024.4

Displaying Information.
Name: Magdalena Dankova
Age: 27
Salary: 1024.4

```

### Structure and Functions

Structure variables can be passed to a function and returned in a similar way as normal arguments.

#### Example:

```

#include <iostream>
using namespace std;

struct Person
{
    char name[50];
    int age;
    float salary;
};

void displayData(Person); // Function declaration

int main()
{
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
}

```

```

    displayData(p);

    return 0;
}

void displayData(Person p)
{
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout << "Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}

```

### Output

```

Enter Full name: Bill Jobs
Enter age: 55
Enter salary: 34233.4

Displaying Information.
Name: Bill Jobs
Age: 55
Salary: 34233.4

```

In this program, user is asked to enter the name, age and salary of a Person inside main() function. Then, the structure variable p is passed to a function using.

```
displayData(p);
```

The return type of displayData() is void and a single argument of type structure Person is passed. Then the members of structure p is displayed from this function.

### Lab Tasks:

#### Task 1

Write a C++ program that maintain record of students. Student contains following details:

- ID
- Name
- Department
- Email
- Phone no

Create a structure of student. Ask user to enter record for 5 students. Store those details in variables of student's type. Print those records on screen.

#### Task 2

Write a C++ program to do the following:

- Set up a structure/record to store products; each product has a name, a model number and a price.
- Choose appropriate types for these fields.

- Your program should contain a loop which allows entry via the keyboard of up to 10 products, to be stored in an array.
- The loop can be stopped by entry of "quit" for the product name
- Your program should include a function to display all the products details after the entry loop has finished.

### **Task 3**

Write a C++ program that compute Net Salary of Employee. Program contains two user defined functions *empSalary()* and *display()*.

- Create a structure of Employee that contains following data members:  
EmployeeNumber, Name, BasicSalary, HouseAllowance, MedicalAllowance, Tax, GrossPay and NetSalary
- Employee number, name and basic salary must be taken input from the user.
- *empSalary()* compute salary with given criteria
  - HouseAllowance = 10% of BasicSalary
  - Medical Allowance = 5% of Basic Salary
  - Tax = 4 % of Basic Salary
  - GrossSalary = Basic+HouseAllowance+MedicalAllowance
  - NetSalary = GrossSalary – Tax
- *display()* for displaying details of Employee

```
#Sample Output
Enter the Employee Number :10129
Enter the employee name: Ahmed Ali
Enter the Basic Salary: 16500

*****
      EMPLOYERS SALARY DETAILS
*****
Employee Number:10129
Employee Name: Ahmed Ali
Basic Salary: 16500
House Allowance:1650
Medical Allowance: 825
Gross Salary: 18975
Tax Deduction: 660
Net Salary: 18315
```

## References

1. [https://www.owl.net.rice.edu/~ceng303/manuals/fortran/FOR3\\_3.html](https://www.owl.net.rice.edu/~ceng303/manuals/fortran/FOR3_3.html)
2. [https://www.comp.nus.edu.sg/~cs1010/4\\_misc/jumpstart/chap3.pdf](https://www.comp.nus.edu.sg/~cs1010/4_misc/jumpstart/chap3.pdf)
3. <https://www.brainkart.com>
4. [docs.microsoft.com](https://docs.microsoft.com)
5. <https://ciphertrick.com>
6. <https://www.programiz.com>
7. <https://www.w3schools.com>
8. <http://web.engr.oregonstate.edu>
9. <https://tutorialink.com>
10. <https://www.geeksforgeeks.org>
11. [elmcp.net](http://elmcp.net)
12. [www.docstoc.com](http://www.docstoc.com)
13. [pastebin.com](https://pastebin.com)
14. [www.kelvinomieno.com](http://www.kelvinomieno.com)
15. [www.geeksforgeeks.org](http://www.geeksforgeeks.org)
16. [www.cs.fsu.edu](http://www.cs.fsu.edu)
17. [www.softwaredynamix.com](http://www.softwaredynamix.com)
18. [www.skar.us](http://www.skar.us)
19. [www.programmingwithbasics.com](http://www.programmingwithbasics.com)
20. [www.coursehero.com](http://www.coursehero.com)
21. [tutorial-cpp-programming.blogspot.com](http://tutorial-cpp-programming.blogspot.com)
22. [www.slideshare.net](http://www.slideshare.net)
23. [fr.slideshare.net](http://fr.slideshare.net)
24. [www.programiz.com](http://www.programiz.com)
25. [idoc.vn](http://idoc.vn)
26. [manualzz.com](http://manualzz.com)
27. [files.eric.ed.gov](http://files.eric.ed.gov)
28. [www.goneboarding.co.uk](http://www.goneboarding.co.uk)
29. [issuu.com](http://issuu.com)
30. [home.comcast.net](http://home.comcast.net)
31. <https://www.cplusplus.com/doc/oldtutorial/files/>
32. <https://www.programiz.com>
33. <https://www.tutorialspoint.com/>
34. <https://www.softwaretestinghelp.com/>
35. <http://www.tutorialdost.com/Cpp-Programming-Tutorial/>