

LAB MANUAL

Course: CSC103-Programming Fundamentals



Department of Computer Science

COMSATS UNIVERSITY ISLAMABAD

List of Lab Tasks

WEEK	Topics	Page no.
Week 1	The Programming Environment	3
Week 2	Arithmetic Operator and Displaying variable values	6
Week 3	Arithmetic Operators	9
Week 4	Output Formatting	11
Week 5	Relational Operators and Decision Control Structure	25
Week 6	Nested if and switch/case structure	35
Week 7	Iterative Structure: While Loop	39
Week 8	Iterative Structure: Do While Loop	43
Week 9	Nested Loops	46
Week 10	Functions	47
Week 11	Functions and Arrays	53
Week 12	2D Arrays	62
Week 13	Pointers and Arrays	69
Week 14	Structures	73
Week 15	Structures with Functions	76
Week 16	File Handling	79

CSC 103 – Programming Fundamentals - Lab

Week1: The Programming Environment

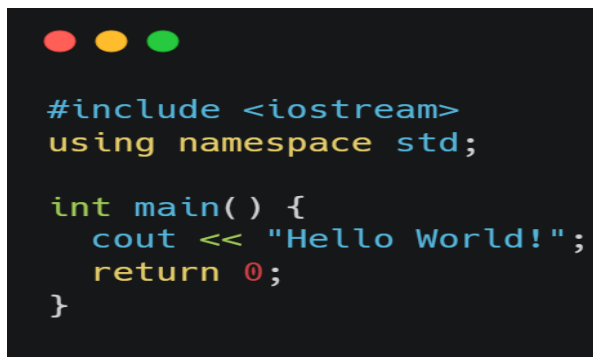
Learning Objectives:

The objectives of this first experiment are:

1. To make you familiar with the Windows programming environment and the "C++" compiler. For this purpose, you will use Code Blocks. You will write, compile and run a (it may be your first) program in "C++"

First C++ Program

Hello World



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

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

C++ Syntax

Line 1: `#include <iostream>` is a **header file library** that lets us work with input and output objects, such as `cout` (used in line 5). Header files add functionality to C++ programs

Line 2: `using namespace std` means that we can use names for objects and variables from the standard library.

Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

Line 4: Another thing that always appear in a C++ program, is `int main()`. This is called a **function**. Any code inside its curly brackets `{ }` will be executed.

Line 5: `cout` (pronounced "see-out") is an **object** used together with the *insertion operator* (`<<`) to output/print text. In our example it will output "Hello World".

Note: Every C++ statement ends with a semicolon `;`

Line 6: `return 0` ends the main function.

Line 7: Do not forget to add the closing curly bracket `}` to actually end the main function.

Omitting namespace

Some C++ programs that runs without the standard namespace library. The `using namespace std` line can be omitted and replaced with the `std` keyword, followed by the `::` operator for some objects:

Example



```
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

C++ Output (Print on Screen)

The `cout` object, together with the `<<` operator, is used to output values/print text:

Note:

You can add as many `cout` objects as you want. However, note that it does not insert a new line at the end of the output:



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

int main() {
    cout << "Hello World!";
    cout << "I am learning C++";
    return 0;
}
```

New Line:

To insert a new line, you can use the `\n` character:

```

#include <iostream>
using namespace std;

int main() {
    cout << "Hello World! \n";
    cout << "I am learning C++";
    return 0;
}

```

Another way to insert a new line, is with the **endl** manipulator:

```

#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
    cout << "I am learning C++";
    return 0;
}

```

LAB Task:

Print following shape using both single and multiple cout statement(s):

1.	2.	3.	4.	5.	6.
*	*****	*	*	*****	*****
***	* *	**	**	*****	*****
*****	* *	***	***	***	***
***	* *	****	****	**	**
*	*****	*****	*****	*	*

CSC 103 – Programming Fundamentals - Lab

Week2: Display Variable Values and Arithmetic Operators

Display Variable Values and Arithmetic Operators

The `cout` object is used together with the `<<` operator to display variables. To combine both text and a variable, separate them with the `<<` operator:

Example:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int myAge=25;
8      cout << "My name is James Taylor."<<endl;
9      cout << "James is " <<myAge<< " years old."<<endl;
10     return 0;
11 }
```

Exercise

1. Create a variable named `myNum`, assign value 50 to it and display it on screen.
2. Create a variable named `z`, assign the sum of values in `x` and `y` that are initialized before. Display the result.
3. Create three variables of the same type i.e. `double`, using the **comma-separated list**. Calculate the sum of three initialized values and display the result.

Example:

Program to calculate the sum of two numbers.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int a,b,sum;
8      cout << "Enter values of a and b: ";
9      cin>>a>>b;
10     sum=a+b;
11     cout << "Sum of " <<a<<" and "<<b<<" is "<<sum<<". ";
12     return 0;
13 }
```

Lab Tasks:

Lab Task 1:

Swap two numbers that are stored in two variables let's say a and b.

Suppose before swapping

a = 25

b = 20

After swapping

a = 20

b = 25

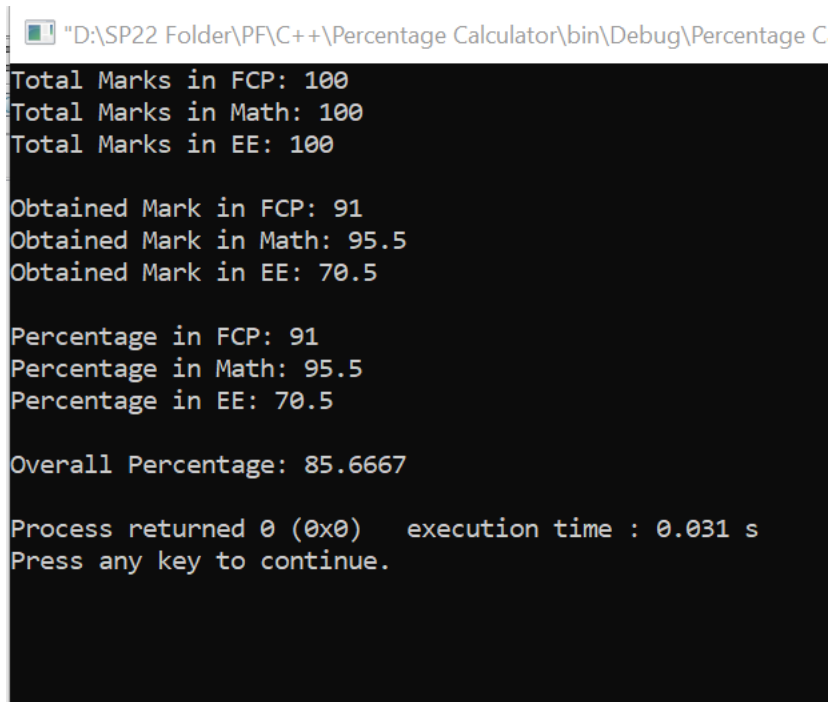
Perform swapping

- Using the third variable
- Without using a third variable.

Lab Task 2:

Write a program which provides total and obtained marks of a student in 3 subjects (say FCP, Math and EE). The program then displays the percentage of marks in each subject and also displays overall percentage.

Sample Output:

A screenshot of a debugger window showing the output of a C++ program. The window title is "D:\SP22 Folder\PF\C++\Percentage Calculator\bin\Debug\Percentage C". The output text is as follows:

```
Total Marks in FCP: 100
Total Marks in Math: 100
Total Marks in EE: 100

Obtained Mark in FCP: 91
Obtained Mark in Math: 95.5
Obtained Mark in EE: 70.5

Percentage in FCP: 91
Percentage in Math: 95.5
Percentage in EE: 70.5

Overall Percentage: 85.6667

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

Lab Task 3:

Write a C++ code which get three numbers from user and find their average.

Lab Task 4:

Write a C++ program which get distance and time from user and calculate the velocity.

Hint: $\text{Velocity} = \text{distance} / \text{time}$

CSC 103 – Programming Fundamentals - Lab

Week 3: Arithmetic Operators

Lab Task 1:

Write a program that prompts the user to input a length expressed in centimetres. The program should then convert the length to inches (to the nearest inch) and output the length expressed in yards, feet, and inches, in that order.

For example, suppose the input for centimetres is 312. To the nearest inch, 312 centimetres is equal to 123 inches. 123 inches would thus be output as:

3 yard(s), 1 foot (foot), and 3 inch(es).

Hint:

1 Inch = 2.54 Centimetres

1 Foot = 12 Inches

1 Yard = 36 Inches

Lab Task 2:

- a) Write a C++ program that prompts the user to input the elapsed time for an event in hours, minutes, and seconds. The program then outputs the elapsed time in seconds.
- b) Write a C++ program that prompts the user to input the elapsed time for an event in seconds. The program then outputs the elapsed time in hours, minutes, and seconds. (For example, if the elapsed time is 9630 seconds, then the output is 2:40:30.)

Lab Task 3:

Write a program that calculates and prints the monthly paycheck for an employee. The net pay is calculated after taking the following deductions:

```
Federal Income Tax: 15%
State Tax: 3.5%
Social Security Tax: 5.75%
Medicare/Medicaid Tax: 2.75%
Pension Plan: 5%
Health Insurance: $75.00
```

Your program should prompt the user to input the gross amount and the employee's name. Format your output to have two decimal places. A sample output follows:

```
Bill Robinson
Gross Amount: ..... $3575.00
Federal Tax: ..... $ 536.25
State Tax: ..... $ 125.13
Social Security Tax: ..... $ 205.56
Medicare/Medicaid Tax: ... $ 98.31
Pension Plan: ..... $ 178.75
Health Insurance: ..... $ 75.00
Net Pay: ..... $2356.00
```

Note that the first column is left-justified, and the right column is right-justified.

CSC 103 – Programming Fundamentals Lab

Week 4: Output Formatting

Output Formatting

Formatting program output is an essential part of any serious application. Surprisingly, most C++ textbooks don't give a full treatment of output formatting. The purpose of this section is to describe the full range of formatting abilities available in C++.

Formatting in the standard C++ libraries is done through the use of *manipulators*, special variables or objects that are placed on the output stream. Most of the standard manipulators are found in `<iostream>` and so are included automatically. The standard C++ manipulators are not keywords in the language, just like `cin` and `cout`, but it is often convenient to think of them as a permanent part of the language.

The standard C++ output manipulators are:

endl

- places a new line character on the output stream. This is identical to placing `'\n'` on the output stream.

```
#include<iostream>
```

```
using namespace std;
```

```
int main()
{
    cout << "Hello world 1" << endl;
    cout << "Hello world 2\n";

    return 0;
}
```

produces

```
Hello world 1
Hello world 2
```

Field width

A very useful thing to do in a program is output numbers and strings in fields of fixed width. This is crucial for reports.

setw()

- Adjusts the field width for the item about to be printed.

Important Point

The `setw()` manipulator only affects the *next* value to be printed.

The `setw()` manipulator takes an integer argument which is the minimum field width for the value to be printed.

Important Point

All manipulators that take arguments are defined in the header file `iomanip`. This header file must be included to use such manipulators.

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

using namespace std;

int main()
{
    // A test of setw()

    cout << "*" << -17 << "*" << endl;
    cout << "*" << setw(6) << -17 << "*" << endl << endl;

    cout << "*" << "Hi there!" << "*" << endl;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;
    cout << "*" << setw(3) << "Hi there!" << "*" << endl;

    return 0;
}
```

Produces

```
*-17*
*   -17*

*Hi there!*
*           Hi there!*
*Hi there!*
```

Note that the values are right justified in their fields. This can be changed. Note also what happens if the value is too big to fit in the field.

Important Point

The argument given to `setw()` is a *minimum* width. If the value needs more space, the output routines will use as much as is needed.

Important Point

The default field width is 0.

Justification

Values can be justified in their fields. There are three manipulators for adjusting the justification: `left`, `right`, and `internal`.

Important Point

The default justification is right justification.

Important Point

All manipulators except `setw()` are persistent. Their effect continues until explicitly changed.

left

- left justify all values in their fields.

right

- right justify all values in their fields. This is the default justification value.

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

using namespace std;

int main()
{
    cout << "*" << -17 << "*" << endl;
    cout << "*" << setw(6) << -17 << "*" << endl;
    cout << left;
    cout << "*" << setw(6) << -17 << "*" << endl << endl;

    cout << "*" << "Hi there!" << "*" << endl;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;
    cout << right;
    cout << "*" << setw(20) << "Hi there!" << "*" << endl;

    return 0;
}

produces
*-17*
*   -17*
*-17   *

*Hi there!*
*Hi there!      *
*              Hi there!*
```

internal

- internally justifies numeric values in their fields. Internal justification separates the sign of a number from its digits. The sign is left justified and the digits are right justified. This is a useful format in accounting and business programs, but is rarely used in other applications.

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

using namespace std;

int main()
{
    cout << setw(9) << -3.25 << endl;
    cout << internal << setw(9) << -3.25 << endl;

    return 0;
}
produces
-3.25
-    3.25
```

boolalpha and noboolalpha

Boolean values (variables of type `bool`) are normally printed as 0 for false and 1 for true. It is sometimes nice to print out these values as words. The `boolalpha` manipulator will print out true and false values as `true` and `false` respectively. The `noboolalpha` manipulator will return the printed values to 1 and 0.

```
#include <iostream>

using namespace std;

int main()
{
    bool bt = true, bf = false;

    cout << "bt: " << bt << " bf: " << bf << endl;
    cout << boolalpha;
    cout << "bt: " << bt << " bf: " << bf << endl;
    cout << noboolalpha;
    cout << "bt: " << bt << " bf: " << bf << endl;

    return 0;
}
produces
bt: 1 bf: 0
bt: true bf: false
bt: 1 bf: 0
```

showpos and noshowpos

This manipulator determines how positive numbers are printed. A negative number is traditionally printed with a minus sign in front. The lack of a minus sign means a positive value. However some accounting and scientific applications traditionally place a plus sign in front of positive numbers just to emphasize the fact that the number is positive. Using the `showpos` manipulator makes this happen automatically. The `noshowpos` manipulator returns the output state to placing nothing in front of positive values.

```
#include <iostream>

using namespace std;

int main()
{
    int pos_int = 4, neg_int = -2, zero_int = 0;
    double pos_f = 3.5, neg_f = -31.2, zero_f = 0.0;

    cout << "pos_int: " << pos_int << " neg_int: " << neg_int;
    cout << " zero_int: " << zero_int << endl;
    cout << "pos_f: " << pos_f << " neg_f: " << neg_f;
    cout << " zero_f: " << zero_f << endl << endl;

    cout << showpos;

    cout << "pos_int: " << pos_int << " neg_int: " << neg_int;
    cout << " zero_int: " << zero_int << endl;
    cout << "pos_f: " << pos_f << " neg_f: " << neg_f;
    cout << " zero_f: " << zero_f << endl << endl;

    return 0;
}
```

produces

```
pos_int: 4  neg_int: -2  zero_int: 0
pos_f: 3.5  neg_f: -31.2  zero_f: 0

pos_int: +4  neg_int: -2  zero_int: +0
pos_f: +3.5  neg_f: -31.2  zero_f: +0
```

Note that zero is considered to be a positive number in this case.

Integer base

Computer science applications occasionally use number systems other than base 10. The most frequently used number systems (after decimal) are octal (base 8) and hexadecimal (base 16). Octal number systems use the digits 0 through 7. Hexadecimal systems use the digits 0 through 9 and add the digits A through F to represent in one digit the values ten through fifteen.

The manipulators `dec`, `oct`, and `hex` change the base that is used to print out integer values.

Important Point

The base used for output formatting is completely independent from how a value is stored in machine memory. During output, the bit patterns in memory are converted to character sequences that are meaningful to human beings.

```
#include <iostream>

using namespace std;

int main()
{
    // These sequences of characters are treated as decimal numbers by
    // the compiler and converted to the machine internal format.

    long int pos_value = 12345678;
    long int neg_value = -87654321;
    float value = 2.71828;

    cout << "The decimal value 12345678 is printed out as" << endl;

    cout << "decimal:      " << pos_value << endl;
    cout << "octal:          " << oct << pos_value << endl;
    cout << "hexadecimal: " << hex << pos_value << endl << endl;

    cout << "The decimal value -87654321 is printed out as" << endl;
    cout << "decimal:      " << dec << neg_value << endl;
    cout << "octal:          " << oct << neg_value << endl;
    cout << "hexadecimal: " << hex << neg_value << endl << endl;

    cout << "The decimal value 2.71828 is printed out as" << endl;
    cout << "decimal:      " << dec << value << endl;
    cout << "octal:          " << oct << value << endl;
    cout << "hexadecimal: " << hex << value << endl << endl;

    return 0;
}
```

produces

```
The decimal value 12345678 is printed out as
decimal:      12345678
octal:        57060516
hexadecimal: bc614e
```

```
The decimal value -87654321 is printed out as
decimal:      -87654321
octal:        37261500117
hexadecimal: fac6804f
```

```
The decimal value 2.71828 is printed out as
decimal:      2.71828
```



```
octal:      2.71828
hexadecimal: 2.71828
```

Note that negative integers are not printed as such in octal or hexadecimal. Rather, the internal bit patterns are interpreted as always being positive values.

Also note that floating point values are always printed out in decimal format.

showbase and noshowbase

When a program is printing out integers in decimal, octal, and hex, it can become difficult to determine at a glance what the real value of a number is. For example, a program outputs the string of characters `"1234"`. Does this represent a decimal 1234, an octal 1234 or a hexadecimal 1234? All are legal but very different internal values.

By tradition in C/C++, any printed integer value that is immediately preceded by a zero is considered to be an octal value. Any printed integer that is immediately preceded by a `"0x"` is considered to be a hexadecimal value. For example the output `1234` is likely a decimal value. The output `01234` is considered to be octal and `0x1234` is considered to be hexadecimal.

By default, these prefixes are not added. The programmer can add them or the `showbase` manipulator can be used. This manipulator simply provides an appropriate prefix to octal and hexadecimal values. The `noshowbase` manipulator turns off this behavior.

```
#include <iostream>

using namespace std;

int main()
{
    long int value = 12345678;

    cout << "The decimal value 12345678 is printed out with showbase as"
    << endl;

    cout << showbase;

    cout << "decimal:      " << value << endl;
    cout << "octal:         " << oct << value << endl;
    cout << "hexadecimal: " << hex << value << endl << endl;

    cout << "The decimal value 12345678 is printed out without showbase"
    << endl;

    cout << noshowbase;

    cout << "decimal:      " << dec << value << endl;
```

```

    cout << "octal:          " << oct << value << endl;
    cout << "hexadecimal: " << hex << value << endl << endl;

    return 0;
}

```

produces

The decimal value 12345678 is printed out with showbase as

```

decimal:      12345678
octal:        057060516
hexadecimal: 0xbc614e

```

The decimal value 12345678 is printed out without showbase as

```

decimal:      12345678
octal:        57060516
hexadecimal: bc614e

```

uppercase and nouppercase

Numeric output consists largely of numeric digits. But there are a few places where alphabetic characters make their appearance. These include the `0x` prefix for hexadecimal numbers, the hexadecimal digits `a` through `f`, and the `e` that is used in scientific notation for large floating point numbers (`6.02e+23`). In some applications, it is important that these letters appear as uppercase letters instead of lowercase. The `uppercase` manipulator enables this behavior. The `nouppercase` manipulator turns all of these characters back to lower case.

```

#include <iostream>

using namespace std;

int main()
{
    long int value = 12345678;
    float value_f = 6020000000.0;

    cout << "Some values without the uppercase manipulator" << endl;
    cout << showbase << hex;
    cout << "hexadecimal: " << value << endl;
    cout << "exponential: " << value_f << endl << endl;

    cout << uppercase;

    cout << "Some values with the uppercase manipulator" << endl;
    cout << "hexadecimal: " << value << endl;
    cout << "exponential: " << value_f << endl << endl;

    return 0;
}

```

produces

Some values without the uppercase manipulator

```

hexadecimal: 0xbc614e
exponential: 6.02e+09

```

Some values with the uppercase manipulator

```
hexadecimal: 0XBC614E
exponential: 6.02E+09
```

Floating Point Output

There are 3 floating point formats: general, fixed, and scientific. Fixed format always has a number, decimal point, and fraction part, no matter how big the number gets, i.e., not scientific notation. 6.02e+17 would be displayed as 6020000000000000000 instead of 6.02e+17.

Scientific format always displays a number in scientific notation. The value of one-fourth would not be displayed as 0.25, but as 2.5e-01 instead.

General format is a mix of fixed and scientific formats. If the number is small enough, fixed format is used. If the number gets too large, the output switches over to scientific format. General format is the default format for floating point values.

fixed and scientific

The manipulator `fixed` will set up the output stream for displaying floating point values in fixed format.

The `scientific` manipulator forces all floating point values to be displayed in scientific notation.

Unfortunately, there is no manipulator to place the output stream back into general format. The author of these notes considers this to be a design flaw in the standard C++ libraries. There is a way to place the output stream back into general format, but it's not pretty and requires more explanation than is appropriate here. In short, here's the magic incantation

```
cout.unsetf(ios::fixed | ios::scientific);
```

In order to use this statement, you need a `using` declaration for the `ios` class.

```
#include <iostream>

using namespace std;

int main()
{
    float small = 3.1415926535897932384626;
    float large = 6.0234567e17;
    float whole = 2.000000000;
```

```

cout << "Some values in general format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

cout << scientific;

cout << "The values in scientific format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

cout << fixed;

cout << "The same values in fixed format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

// Doesn't work -- doesn't exist
// cout << general;

cout.unsetf(ios::fixed | ios::scientific);

cout << "Back to general format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

return 0;
}

```

produces

Some values in general format

```

small:  3.14159
large:  6.02346e+17
whole:  2

```

The values in scientific format

```

small:  3.141593e+00
large:  6.023457e+17
whole:  2.000000e+00

```

The same values in fixed format

```

small:  3.141593
large:  602345661202956288.000000
whole:  2.000000

```

Back to general format

```

small:  3.14159
large:  6.02346e+17
whole:  2

```

`setprecision()`

An important point about floating point output is the *precision*, which is roughly the number of digits displayed for the number. The exact definition of the precision depends on which output format is currently being used.

In general format, the precision is the maximum number of digits displayed. This includes digits before and after the decimal point, but does not include the decimal point itself. Digits in a scientific exponent are not included.

In fixed and scientific formats, the precision is the number of digits after the decimal point.

Important Point

The default output precision is 6.

The `setprecision` manipulator allows you to set the precision used for printing out floating point values. The manipulator takes an integer argument. The header file `<iomanip>` must be included to use this manipulator.

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

using namespace std;

int main()
{
    // Testing setprecision

    float small = 3.1415926535897932384626;
    float large = 6.0234567e17;
    float whole = 2.000000000;

    cout << "Some values in general format (default precision: 6)" <<
endl;
    cout << "small: " << small << endl;
    cout << "large: " << large << endl;
    cout << "whole: " << whole << endl << endl;

    cout << "Some values in general format (with setprecision(4))" <<
endl;
    cout << setprecision(4);
    cout << "small: " << small << endl;
    cout << "large: " << large << endl;
    cout << "whole: " << whole << endl << endl;
```

```

    cout << scientific;

    cout << "The values in scientific format (default precision: 6)" <<
endl;
    cout << setprecision(6);
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    cout << "The values in scientific format with setprecision(4)" <<
endl;
    cout << setprecision(4);
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    cout << fixed;

    cout << "The same values in fixed format (default precision: 6)" <<
endl;
    cout << setprecision(6);
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    cout << "The same values in fixed format with setprecision(4)" <<
endl;
    cout << setprecision(4);
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    // Doesn't work -- doesn't exist
    // cout << general;

    cout.unsetf(ios::fixed | ios::scientific);

    cout << "Back to general format (with setprecision(4))" << endl;
    cout << "small:  " << small << endl;
    cout << "large:  " << large << endl;
    cout << "whole:  " << whole << endl << endl;

    return 0;
}

```

Produces the following output:

```

Some values in general format (default precision: 6)
small:  3.14159
large:  6.02346e+017
whole:  2

Some values in general format (with setprecision(4))
small:  3.142
large:  6.023e+017

```

```

whole:  2

The values in scientific format (default precision: 6)
small:  3.141593e+000
large:  6.023457e+017
whole:  2.000000e+000

The values in scientific format with setprecision(4)
small:  3.1416e+000
large:  6.0235e+017
whole:  2.0000e+000

The same values in fixed format (default precision: 6)
small:  3.141593
large:  602345661202956288.000000
whole:  2.000000

The same values in fixed format with setprecision(4)
small:  3.1416
large:  602345661202956288.0000
whole:  2.0000

Back to general format (with setprecision(4))
small:  3.142
large:  6.023e+017
whole:  2

```

showpoint and noshowpoint

There is one aspect of printing numbers in general format that is either very nice or very annoying depending on your point of view. When printing out floating point values, only as many decimal places as needed (up to the precision) are used to print out the values. In other words, trailing zeros are not printed. This is nice and compact, but impossible to get decimal points to line up in tables.

The `showpoint` manipulator forces trailing zeros to be printed, even though they are not needed. By default this option is off. As can be seen from previous examples, this manipulator is not needed in fixed or scientific format, only in general format.

```

#include <iostream>

using namespace std;

int main()
{
    float lots = 3.1415926535;
    float little1 = 2.25;
    float little2 = 1.5;
    float whole = 4.00000;

    cout << "Some values with noshowpoint (the default)" << endl << endl;

```

```

cout << "lots:      " << lots << endl;
cout << "little1: " << little1 << endl;
cout << "little2: " << little2 << endl;
cout << "whole:    " << whole << endl;

cout << endl << endl;

cout << "The same values with showpoint" << endl << endl;

cout << showpoint;

cout << "lots:      " << lots << endl;
cout << "little1: " << little1 << endl;
cout << "little2: " << little2 << endl;
cout << "whole:    " << whole << endl;

return 0;
}

```

produces

Some values with noshowpoint (the default)

```

lots:      3.14159
little1: 2.25
little2: 1.5
whole:    4

```

The same values with showpoint

```

lots:      3.14159
little1: 2.25000
little2: 1.50000
whole:    4.00000

```

Home Task: Explore setprecision and showpoint, when used together in different modes (general, fixed and scientific).

CSC 103 – Programming Fundamentals Lab

Week 5: Relational Operators and Decision Control Structure

Relational and Equality Operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C:

Expression	Is true if
$x = y$	x is equal to y
$x \neq y$	x is not equal to y
$x > y$	x is greater than y
$x < y$	x is less than y
$x \geq y$	x is greater than or equal to y
$x \leq y$	x is less than or equal to y

Logical operators (!, &&, ||)

The ‘NOT’ Operator: The Operator ! is the C++ operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

1	!(5 == 5) // evaluates to false because the expression at its right (5 == 5) is true.
2	!(6 <= 4) // evaluates to true because (6 <= 4) would be false.
3	!true // evaluates to false
4	!false // evaluates to true.

The logical operators && and || are used when evaluating two expressions to obtain a single relational result.

The ‘AND’ Operator: The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise.

The ‘OR’ Operator: The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves.

For example:

1	((5 == 5) && (3 > 6)) // evaluates to false (true && false).
2	((5 == 5) (3 > 6)) // evaluates to true (true false).

When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest. Therefore, in this last example ((5 == 5) || (3 > 6)), C++

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

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

Forms of *if*:

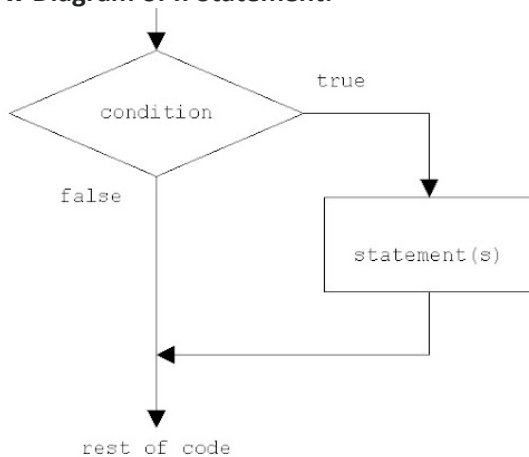
If Statement:

The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false, then the statements inside “if” are skipped.

Syntax:

```
if (condition) {  
  //body of if  
}
```

Flow Diagram of If Statement:



Example 1:

Write a program to take two numbers from the user and check whether a first number is smaller than the second number or not.

Code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int num1, num2;
8      cout<<"Enter two numbers : ";
9      cin>>num1>>num2;
10     if (num1 < num2)
11         cout << num1 <<" is smaller than " << num2 <<endl;
12     return 0;
13 }
```

Output:

```
Enter two numbers : 4 5
4 is smaller than 5

Process returned 0 (0x0)   execution time : 4.205 s
Press any key to continue.
```

Example2: Use of Multiple If statements:

We can use multiple if statements to check more than one condition.

Write a program to take two numbers as input and check whether a first number is smaller, greater or equal. Use only if statement for this purpose.

Code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int num1, num2;
8      cout<<"Enter two numbers : ";
9      cin>>num1>>num2;
10     if (num1 < num2)
11         cout << num1 <<" is smaller than "<< num2 <<endl;
12     if (num1 > num2)
13         cout << num1 <<" is larger than "<< num2 <<endl;
14     if (num1 == num2)
15         cout << num1 <<" is equal "<< num2 <<endl;
16     return 0;
17 }
```

Output:

```
Enter two numbers : 77 77
77 is equal 77

Process returned 0 (0x0)   execution time : 3.833 s
Press any key to continue.
```

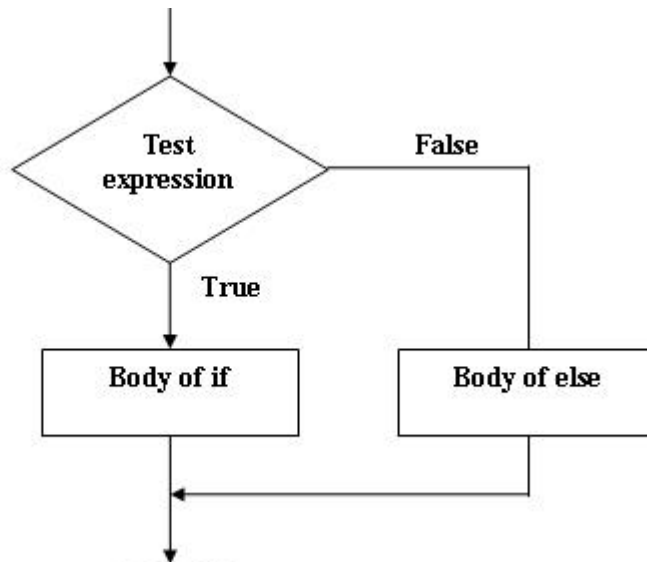
If Else Statement:

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” statements are skipped. If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

Syntax:

```
If (condition) {  
  // statement inside body of if  
}  
Else {  
  // statement inside body of else  
}
```

Flow Diagram of if else statement:



Example 3:

Write a program which will take age of a person from user and decide whether a person is eligible to vote or not. The person must be greater than or equal to 18 years to be eligible to vote.

Code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int age;
8      cout << "Enter age of a person : ";
9      cin >> age;
10     if (age > 18)
11     {
12         cout<<"The person is eligible for voting."<<endl;
13     }
14     else
15     {
16         cout<<"The person is not eligible for voting."<<endl;
17     }
18     return 0;
19 }
```

Output:

```
Enter age of a person : 20
The person is eligible for voting.

Process returned 0 (0x0)   execution time : 1.991 s
Press any key to continue.
```

Note: If there is **only one statement** is present in the “if” or “else” body then you do not need to use the braces. For example, the above program can be rewritten like this:

Code:

```
#include <iostream>

using namespace std;

int main()
{
    int age;
    cout << "Enter age of a person : ";
    cin >> age;
    if (age > 18)
        cout<<"The person is eligible for voting."<<endl;
    else
        cout<<"The person is not eligible for voting."<<endl;
    return 0;
}
```

Output:

```
Enter age of a person : 16
The person is not eligible for voting.

Process returned 0 (0x0)   execution time : 3.102 s
Press any key to continue.
```

Nested If Else Statement:

When an if else statement is present inside the body of another “if” or “else” then this is called nested if else.

Syntax:

```
if(condition) {
    //Nested if else inside the body of "if"
    if(condition2) {
        //Statements inside the body of nested "if"
    }
    else {
        //Statements inside the body of nested "else"
    }
}
else {
    //Statements inside the body of "else"
}
```

Example 4:

Write a program to take two numbers as input and check whether a first number is smaller, greater or equal. Use Nested if-else statement.

Code:

```
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    cout<<"Enter two numbers : ";
    cin>>num1>>num2;
    if (num1 != num2)
    {
        cout << num1 <<" is not equal to "<< num2 <<endl;
        if(num1 < num2)
            cout << num1 <<" is smaller than "<< num2 <<endl;
        else
            cout << num1 <<" is larger than "<< num2 <<endl;
    }
    else
        cout << num1 <<" is equal to "<< num2 <<endl;

    return 0;
}
```

Output:

```
Enter two numbers : 56 78
56 is not equal to 78
56 is smaller than 78

Process returned 0 (0x0)   execution time : 5.200 s
Press any key to continue.
```

Else-if Statement:

The else if statement is useful when you need to check multiple conditions within the program, nesting of if-else blocks can be avoided using else if statement.

Syntax:

```
if (condition1)
{
    //These statements would execute if the condition1 is true
}
else if(condition2)
{
    //These statements would execute if the condition2 is true
}
```



```

else if (condition3)
{
    //These statements would execute if the condition3 is true
}
.
.
else
{
    //These statements would execute if all the conditions return
false.
}

```

Example 5:

```

#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    cout<<"Enter two numbers : ";
    cin>>num1>>num2;
    if (num1 != num2)
        cout << num1 <<" is not equal to " << num2 <<endl;
    else if(num1 < num2)
        cout << num1 <<" is smaller than " << num2 <<endl;
    else if(num1 > num2)
        cout << num1 <<" is larger than " << num2 <<endl;
    else
        cout << num1 <<" is equal to " << num2 <<endl;

    return 0;
}

```

Output:

```

Enter two numbers : 45 67
45 is not equal to 67

Process returned 0 (0x0)   execution time : 2.846 s
Press any key to continue.

```

Important Points:

1. else and else..if are optional statements, a program having only “if” statement would run fine.
2. else and else..if cannot be used without the “if”.
3. There can be any number of else..if statement in a if else..if block.
4. If none of the conditions are met then the statements in else block gets executed.
5. Just like relational operators, we can also use logical operators such as AND (&&), OR(||) and NOT(!).

TASKS:

1. Write a C++ program to determine whether a number is even or odd using if statement and modulus operator.
2. Take from user time in 24-hour format and convert to 12 hour and display.
 - Declare and initialize 3 variables for hour, minutes and seconds in 24 hour format as Thour, Tmin, Tsec
 - Declare and initialize three variables for hours minutes and seconds in 12 hour format as thour, tmin, tsec.
 - Get user values for Thour, Tmin, Tsec.
 - If Thour greater than 12, subtract 12 from Thours and store in thour. Copy tmin and tsec to Tmin and Tsec respectively. Display the 12 hour values.
 - If Thour is lesser than or equal to 12, copy (24 hour) values to 12 hour variables and display.
3. Write a C++ program, which inputs 3 number x, y and z. The program should then output the sum of positive integers.
4. Write a C++ program to check whether a triangle is valid or not, when the three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all the three angles is equal to 180 degrees.
5. Write a C++ program that inputs three different integers from user, then prints the sum, the average, the product, the smallest and the largest of these numbers.
6. Write a C++ program to check whether a character is an uppercase or lowercase alphabet.
7. **Program Name: (The number-word program 0-9) Hint:** Apply if-else-if (or nested if-else)

Write a C++ program which inputs a one-digit number from the user (i.e. 0-9). The program should then print that number in words, e.g. "Zero" for 0, "One" for 1, "Two" for 2, and so on. If the user does not enter a one-digit number, then program should display an error: "Invalid number!"

Output

```
Enter a one-digit integer(0-9): 5
Five.
```

CSC 103 – Programming Fundamentals - Lab

Week 6: Nested If and Switch/Case Structure

Switch-Case Statement:

Use the switch statement to select one of many code blocks to be executed. Test values of a variable and compares it with multiple cases. Once the case is matched, a block of statement associated with that particular case is executed.

If a case is not matched, the default statement is executed and the control goes out of the switch block

Syntax:

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

This is how it works:

- The **switch** expression is evaluated once
- The value of the expression is compared with the values of each **case**
- If there is a match, the associated block of code is executed
- The **break** and **default** keywords are optional, and will be described later in the tutorial

Why Need Switch-Case?

The complexity of a program increases whenever the number of the alternative path increases. If multiple if-else statements are used, the program might get difficult to read.

Rules for Switch-Case?

- An expression must always execute to a result
- Case labels must be constant and unique
- Case labels must end with a colon
- break keyword must be present for each case
- There can be only one default label

- We can nest multiple switch-case statements

Example:

The example below uses the weekday number to calculate the weekday name:

Code:

```
#include <iostream>

using namespace std;

int main()
{
    int day;
    cout << "Enter day : ";
    cin >> day;
    switch(day)
    {
        case 1:
            cout << "Monday";
            break;
        case 2:
            cout << "Tuesday";
            break;
        case 3:
            cout << "Wednesday";
            break;
        case 4:
            cout << "Thursday";
            break;
        case 5:
            cout << "Friday";
            break;
        case 6:
            cout << "Saturday";
            break;
        case 7:
            cout << "Sunday";
            break;
        default:
            cout << "Invalid Choice";
    }
    return 0;
}
```

The break Keyword:

When C++ reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing. A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default keyword:

The **default** keyword specifies some code to run if there is no case match:

Multiple Cases in Switch-Case

The switch case statement consists of a series of labels, an optional default case, and the statement to execute for each case.

Example

Using a switch-case statement, write a program to take a month from the user and display the number of days each month has. Assume that February has only 28 days (ignore leap year).

Nested Switch-Case

A switch-case statement inside another switch-case statement is termed a nested switch-case.

Syntax:

```
switch(expression 1) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
  
    case z:  
        // code block  
        switch(expression 2) {  
            case a:  
                // code block  
                break;  
            case b:  
                // code block  
                break;  
        }  
    default:  
        // code block  
}
```

LAB TASKS:

1. Write a C++ program that inputs four different integers from user, then prints the sum, the average, the product, the smallest and the largest of these numbers. Use selection statement.
2. Write a C++ program to check whether a character is an uppercase or lowercase or digit alphabet. Using the selection statement.
3. Write a C++ program that take an integer from the user and check whether an integer is positive, negative, or zero using switch case statement.
4. Write a C++ program to take the value from the user as input any alphabet and check whether it is vowel or consonant. Using the switch statement.
5. Write a program which performs arithmetic operations on integers using switch. The user should enter the two integer values and the arithmetic operator.
 - a) Addition (+)
 - b) Subtraction (-)
 - c) Division (/)
 - d) Multiplication (*)
 - e) Modulus (%)
6. Write a C++ program which to find the grace marks for a student using switch. The user should enter the grade obtained by the student and the number of subjects he has failed in.
 - If the student gets A grade and the number of subjects, he failed in is greater than 3, then he does not get any grace. If the number of subjects he failed in is less than or equal to 3 then the grace is of 5 marks per subject.
 - If the student gets B grade and the number of subjects, he failed in is greater than 2, then he does not get any grace. If the number of subjects he failed in is less than or equal to 2 then the grace is of 4 marks per subject.
 - If the student gets C grade and the number of subjects, he failed in is greater than 1, then he does not get any grace. If the number of subjects he failed in is equal to 1 then the grace is of 5 marks per subject

CSC 103 – Programming Fundamentals - Lab

Week 7: While Loop

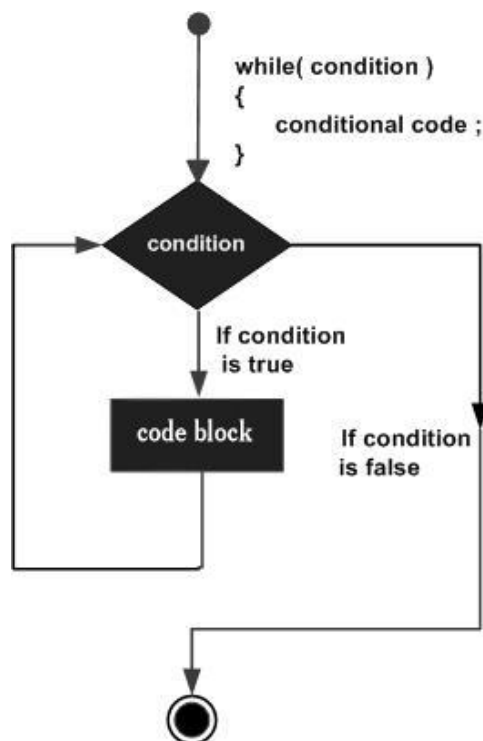
While Loop:

While Loops are used when we don't know how many times certain instructions will be executed. While loop terminates as soon as the condition gets false.

Syntax:

```
while (condition) {  
    // body of while loop  
}
```

Flowchart:



Note: A while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

For Loop:

When you know exactly how many times you want to loop through a block of code, use the **for** loop instead of a **while** loop:

Syntax:

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example

```
for (int i = 0; i < 5; i++) {  
    cout << i << "\n";  
}
```

Example Explanation:

Statement 1 sets a variable before the loop starts (`int i = 0`).

Statement 2 defines the condition for the loop to run (`i` must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.

Statement 3 increases a value (`i++`) each time the code block in the loop has been executed.

TASKS:

Important Note: For all lab tasks, First identify the loop counter variable and the steps to repeat in the each of the following problems:

1. Write a C++ program that reads positive numbers from the user until user enters the -1. The program also determines and displays the sum and average of the numbers. (Using while loop).

Sample Output:

```
Enter a positive integer: 9  
Enter a positive integer: 8  
Enter a positive integer: 7  
Enter a positive integer: 12  
Enter a positive integer: 5  
Enter a positive integer: 17  
Enter a positive integer: -1
```

Sum is 58.

Average is 9.67.

2. The factorial is calculated by multiplying the original number by all the positive integers smaller than itself. Thus, the factorial of 5 is $5*4*3*2*1 = 120$. Write a C++ program to calculate factorial, of any number given by user. (Using while loop).

Sample Output:

Enter a positive integer: 7

Factorial of 7 is 5040.

3. Write a C++ program that inputs from user an integer and prints to the screen the table of that number up to 10 multiples. (Implement this program by using while loop)
4. Write a C++ program that inputs from user an integer and prints to the screen the table of that number up to 10 multiples. (Implement this program by using for loop)

Sample Output (of Lab task 3 and Lab task 4):

Enter a positive integer: 5

**5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50**

5. Write a C++ program that read a positive integer value, and compute the following sequence: If the number is even, halve it; if it's odd, multiply by 3 and add 1. Repeat this process until the value is 1, printing out each value. Finally, print out how many of these operations you performed.

Sample Output:

Enter initial value is 7

**Next value is 22
Next value is 11
Next value is 34
Next value is 17
Next value is 52
Next value is 26
Next value is 13
Next value is 40
Next value is 20
Next value is 10
Next value is 5
Next value is 16
Next value is 8**

```
Next value is    4
Next value is    2
Final value 1, number of steps 16
```

If the input value is less than 1, print a message containing the word `Error` and exit from the program.

CSC 103 – Programming Fundamentals - Lab

Week 8: Iterative Structure

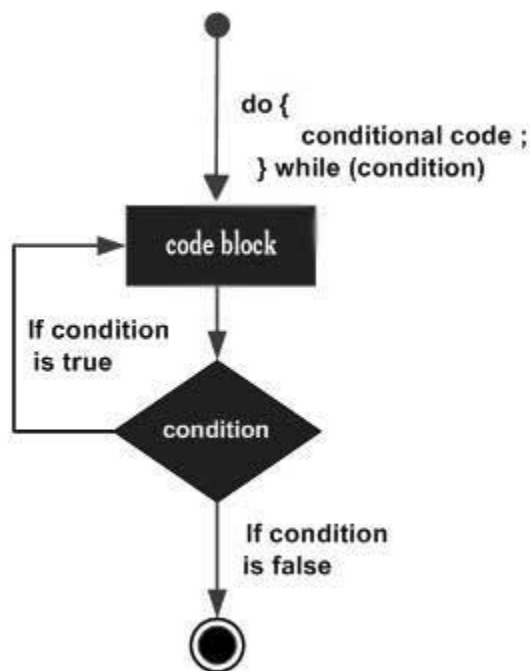
Do-While Loop:

Do while loop is similar to while loop with one exception that it executes the statements inside the body of do-while before checking the condition. On the other hand, in the while loop, first the condition is checked and then the statements in while loop is executed.

Syntax:

```
do {  
  // body  
} while (condition);
```

Flowchart:



LAB TASKS:

Important Note: For all lab tasks, identify the loop counter variable and the steps to repeat in the each of the following problems. Also identify the loop termination condition.

1. Write a C++ program to get numbers from user until the user enters number -999. The program should display the count of positive, negative, even, odd and zeros entered by user.

Sample Output:

```
Enter a number (-999 to stop): 5
Enter a number (-999 to stop): 10
Enter a number (-999 to stop): -7
Enter a number (-999 to stop): 0
Enter a number (-999 to stop): -9
Enter a number (-999 to stop): 0
Enter a number (-999 to stop): 11
Enter a number (-999 to stop): 20
Enter a number (-999 to stop): -999
```

```
# of positive numbers: 4
# of negative numbers: 2
# of zeros: 2
# of odd numbers: 4
# of even numbers: 4
```

2. Write a C++ program that reads a number **N** from user. The value of **N** is between 2 and 10 only. The program then input **N** numbers from user, counts the number of odd and even. finds the sum of the even and odd integers. The number values should be between 4 and 10 only.

Sample Output:

```
Enter a value for N: 1
Error! Invalid value
Enter a value for N: 11
Error! Invalid value
Enter a value for N: 5
Enter a number: 3
Error! Invalid Number
Enter a number: 5
Enter a number: 10
Enter a number: 7
Enter a number: 6
Enter a number: 9
```

```
# of odd numbers: 3
# of even numbers: 2
Sum of odd numbers: 21
```

Sum of even numbers: 16

3. Write a C++ program which repeatedly inputs a pair of positive numbers from the user and prints their sum repeatedly. This process continues until a pair of numbers is reached in which the first integer evenly divides by second integer. In the end, the program should display the total numbers entered.

Sample Output:

Input a pair of integers: 5 3

Sum is: 8

Input a pair of integers: 10 18

Sum is: 28

Input a pair of integers: 20 5

The loop is ended.

Total integers entered: 6

CSC 103 – Programming Fundamentals - Lab

Week 09: Nested Loops

Nested Loops

Tasks:

1. (*Triangle Printing Program*) Write a program that prints the following patterns separately, one below the other. Use for loops to generate the patterns. All asterisks (*) should be printed by a single print statement of the form `cout<< "*" ;` (this causes the asterisks to print side by side).

[*Hint:* The last two patterns are tricky. They require that each line begin with an appropriate number of blanks, i.e., 2 inner loops inside the outer loop; the first will control the number of spaces to print using dynamic condition/initialization and the second will control the number of asterisks to print using dynamic condition/initialization.]



2. The program takes the number input from between 1 to 9. Use nested loops to print numbers such that the outer loop prints the number of rows and the inner loop prints the number of columns. With each increasing row, the number of columns increases. The output looks as follows.

```

1
12
123
1234
12345
123456
1234567
12345678
123456789

```

CSC 103 – Programming Fundamentals - Lab

Week 10: Functions

Statement Purpose:

This lab will give you practical implementation of different types of **user-defined functions**.

Activity Outcomes:

This lab teaches you the following topics:

- How to define own functions
- How to use user-defined functions
- Passing different types of arguments

Introduction:

It is usually a better approach to divide a large code in small functions. A function is a small piece of code used to perform a specific purpose. functions are defined first then called whenever needed. A program may have as many functions as required. Similarly, a function may be called as many times as required.

How to Define Methods

Functions are defined as below.

```
returnType functionName(dataType par1, dataType par2, ...)
{
    statement1;
    statement2;
    ...
    return value;
}
```

A function may or may not return a value. If a function does not return a value, then the return type must be **void** otherwise you need to return the data type of the value to be returned.

A function must have an appropriate name and a pair of parentheses. In parenthesis, you need to mention that whether a function will accept or not any parameters. If a function is not accepting any parameter, then the parenthesis must be left blank otherwise you need to mention the name and type of parameters to be accepted.

A function must return a value if the return type is not **void**.

How to Call A Function

Once a function is defined then it can be called by using its name and providing values to the parameters. The following activities demonstrate that how functions are defined and called in C++.

Activity 1:

Define a function to accept an integer value and return its factorial.

Solution:

```
#include <iostream>

using namespace std;

int fact(int n);

int main()
{
    int num;
    cout << "Enter an integer value : ";
    cin >> num;
    cout << "Factorial of num is "<<fact(num)<<". ";
    return 0;
}

int fact(int n)
{
    int factorial=1;
    for (int i=n; i>=1; i--)
        factorial=factorial*i;
    return factorial;
}
```

Activity 2:

Write a function to accept 2 integer values from user and return their sum.

Solution:

```
#include <iostream>

using namespace std;

int sum(int n, int m);

int main()
{
    int num1, num2;
    cout << "Enter an integer 1 value: ";
    cin >> num1;
    cout << "Enter an integer 2 value: ";
    cin >> num2;
    cout << "Sum of both integers are "<<sum(num1,num2)<<". ";
    return 0;
}

int sum(int n, int m)
{
    int sum = n+m;
    return sum;
}
```

Activity 3:

Define a function to accept an integer value from user and check that whether the given value is prime number or not. If the given value is a prime number, then it will return true otherwise false.

Solution:

```
#include <iostream>
using namespace std;
bool isPrime(int n);
int main()
{
    int num;
    cout << "Enter an integer value: ";
    cin >> num;
    if(isPrime(num)==true)
        cout << "The given number is prime.";
    else
        cout << "The given number is not prime.";
    return 0;
}
```

```
bool isPrime(int n)
{
    bool prime=true;
    for (int i=2;i<n/2;i++)
    {
        int r=n%i;
        if (r==0)
        {
            prime=false;
            break;
        }
    }
    return prime;
}
```

Activity 4:

Define a function to accept an integer value and display its numeric table.

Solution:

```
#include <iostream>
using namespace std;
void table(int n);
int main()
{
    int num;
    cout << "Enter an integer value: ";
    cin >> num;
    table(num);
    return 0;
}

void table(int n)
{
    for (int i=1;i<=10;i++)
        cout << n << " * " << i << " = " << n*i << endl;
}
```

TASKS:

1. Define a function **hypotenuse** that calculates the hypotenuse of a right triangle when the other two sides are given. The function should take two double arguments and return the hypotenuse as a double. Use this function in a program to determine the hypotenuse for each of the triangles shown below.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

Let **a** = side of right triangle
 b = side of right triangle
 c = hypotenuse
Formula: $c^2 = a^2 + b^2$

2. Write a function to accept marks of a student (between 0-100) and return the grade according to the following criteria.

0 - 49	F
50 - 60	E
61 - 70	D
71 - 80	C
81 - 90	B
91 - 100	A

3. (*Square of Asterisks*) Write a function that displays at the left margin of the screen a solid square of asterisks whose side is specified in integer parameter side. For example, if side is 4, the function displays the following:

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

4. Write a function, which inputs two characters from the user and compares them using a function named 'compareChars'. The function works as follows.
The 'compareChars' function receives 2 Input characters c1 and c2 as formal parameters, and returns following values:
- Returns 1, if c1 is larger than c2 character in alphabetical order.
 - Returns -1, if c1 is smaller than c2 character in alphabetical order.
 - Returns 0, if c1 is equal to c2.
 - Returns -99, if either c1 or c2 is not an alphabet.

(Hint: the characters are compared using their ASCII values).

Note: The function compareChar should ignore the alphabet's case, i.e., 'n' and 'N' must be treated as the same.

Call this function inside main using different characters as actual arguments and verify the function output by displaying its returned value on screen.

CSC 103 – Programming Fundamentals - Lab

Week 11: Functions and Arrays

Functions (Value and Reference Parameter)

Activity 1: Calculate Grade

The following program takes a course score (a value between 0 and 100) and determines a student's course grade. This program has three functions: **main**, **getScore**, and **printGrade** as follows:

1. **main**
 - a. Get the course score.
 - b. Print the course grade.
2. **getScore**
 - a. Prompt the user for the input.
 - b. Get the input.
 - c. Print the course score.
3. **printGrade**
 - a. Calculate the course grade.
 - b. Print the course grade.

The complete program is as follows:

```
// Example 7-5: This program reads a course score and prints the
//associated //course grade.
#include <iostream>

using namespace std;

void getScore(int& score);
void printGrade(int score);

int main()
{
    int courseScore;
    cout << "Based on the course score, \n" << " this program computes
    the course grade." << endl;

    getScore(courseScore);
    printGrade(courseScore);

    return 0;
}

void getScore(int& score)
{
```

```

cout << "Enter course score: ";
cin >> score;
cout << endl << "Course score is " << score << endl;
}
void printGrade(int cScore)
{
cout << "Your grade for the course is ";
if (cScore >= 90)
    cout << "A." << endl;
else if (cScore >= 80)
    cout << "B." << endl;
else if (cScore >= 70)
    cout << "C." << endl;
else if (cScore >= 60)
    cout << "D." << endl;
else
    cout << "F." << endl;
}

```

Sample Output:

```

Based on the course score,
this program computes the course grade.
Enter course score: 85

```

```

Course score is 85
Your grade for the course is B.

```

Activity 2: Value and Reference Parameter

The following program shows how reference and value parameters work.

//Example 7-6: Reference and value parameters

```

#include <iostream>

using namespace std;

void funOne(int a, int& b, char v);
void funTwo(int& x, int y, char& w);

int main()
{
int num1, num2;
char ch;
num1 = 10; //Line 1
num2 = 15; //Line 2
ch = 'A'; //Line 3
cout << "Line 4: Inside main: num1 = " << num1

```

```

<< ", num2 = " << num2 << ", and ch = " << ch << endl; //Line 4

funOne(num1, num2, ch); //Line 5

cout << "Line 6: After funOne: num1 = " << num1 << ", num2 = "
<< num2 << ", and ch = " << ch << endl; //Line 6

funTwo(num2, 25, ch); //Line 7

cout << "Line 8: After funTwo: num1 = " << num1 << ", num2 = "
<< num2 << ", and ch = " << ch << endl; //Line 8
return 0;
}

void funOne(int a, int& b, char v)
{
    int one;
    one = a; //Line 9
    a++; //Line 10
    b = b * 2; //Line 11
    v = 'B'; //Line 12
    cout << "Line 13: Inside funOne: a = " << a << ", b = " << b <<
    ", v = " << v << ", and one = " << one << endl; //Line 13
}

void funTwo(int& x, int y, char& w)
{
    x++; //Line 14
    y = y * 2; //Line 15
    w = 'G'; //Line 16
    cout << "Line 17: Inside funTwo: x = " << x << ", y = " << y <<
    ", and w = " << w << endl; //Line 17
}

```

Sample Output:

```

Line 4: Inside main: num1 = 10, num2 = 15, and ch = A
Line 13: Inside funOne: a = 11, b = 30, v = B, and one = 10
Line 6: After funOne: num1 = 10, num2 = 30, and ch = A
Line 17: Inside funTwo: x = 31, y = 50, and w = G
Line 8: After funTwo: num1 = 10, num2 = 31, and ch = G

```

Arrays

Statement Purpose:

In this lab, students will know about the basic concepts of Array, storing different types of data in arrays, creating arrays of integers, double and passing arrays to the functions.

Activity Outcomes:

This lab teaches you the following topics:

- Create Arrays
- Accessing indexes/location of variables in arrays
- Accessing maximum, minimum numbers in arrays
- Different array's operation
- Passing arrays to functions

Introduction:

Often you will have to store a large number of values during the execution of a program. Suppose, for instance, that you need to read 100 numbers, compute their average, and find out how many numbers are above the average. Your program first reads the numbers and computes their average, then compares each number with the average to determine whether it is above the average. In order to accomplish this task, the numbers must all be stored in variables. You have to declare 100 variables and repeatedly write almost identical code 100 times. Writing a program this way would be impractical. So, how do you solve this problem?

An efficient, organized approach is needed. C++ and most other high-level languages provide a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. In the present case, you can store all 100 numbers into an array and access them through a single array variable.

Array is a data structure that represents a collection of the same types of data.

The array elements are accessed through the index. The array indices are 0-based, i.e., it starts from 0 to **ARRAY_SIZE-1**.

• Declaring arrays:

```
dataType arrayName[intExp];
```

The element type can be any data type, and all elements in the array will have the same data type. For example, the following code declares a variable `myList` that references an array of double elements.

```
double myList[10];
```

Activity 1: (Processing One-Dimensional Arrays)

This program demonstrates the basic operations on a one-dimensional array:

- Initializing
 - Inputting data
 - Outputting data stored in an array
 - Finding the largest element
- Each operation requires ability to step through elements of the array
 - Easily accomplished by a loop

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

using namespace std;

int main()
{
    double sales[10];
    int index, maxIndex;
    double largestSale, sum, average;

    cout<<fixed<<setprecision(2);

    //Initializing an array
    //The following loop initializes every component of the array
    //sales to 0.0.
    for (index = 0; index < 10; index++)
        sales[index] = 0.0;

    //Reading data into an array
    cout<<"Enter values in array: ";
    for (index = 0; index < 10; index++)
        cin >> sales[index];

    //Printing an array
    cout<<"Values in array are: ";
    for (index = 0; index < 10; index++)
        cout << sales[index] << " ";

    //Finding the sum and average of an array
    sum = 0;
    for (index = 0; index < 10; index++)
        sum = sum + sales[index];
    average = sum / 10;
    cout <<"\nAverage is : "<<average;

    //Largest element in the array
    maxIndex = 0;
```

```

    for (index = 1; index < 10; index++)
        if (sales[maxIndex] < sales[index])
            maxIndex = index;
    largestSale = sales[maxIndex];
    cout << "\nThe largest element is : "<< largestSale;

    return 0;
}

```

Activity 2: (Arrays as Parameters to Functions)

In C++, arrays are passed by reference only. Because arrays are passed by reference only, you do not use the symbol & when declaring an array as a formal parameter.

This example shows how to write functions for array processing and declare an array as a formal parameter.

```

#include <iostream>
using namespace std;
const int ARRAY_SIZE = 10;
//Function Prototype
void fillArray(int arr[],int sizeX);
void printArray(const int arr[],int sizeX);
int sumArray(const int arr[],int sizeX);
int indexLargestElement(const int arr[],int sizeX);

int main()
{
    int list[ARRAY_SIZE] = {0}; //Declare the array list
                                //of 10 components and
                                //initialize each component to 0.

    cout << "list elements: "; //Output the elements of list using
                                //the function printArray
    printArray(list, ARRAY_SIZE);
    cout << endl;

    cout << "Enter " << ARRAY_SIZE << " integers: ";

    //Input data into list using the function fillArray
    fillArray(list, ARRAY_SIZE);
    cout << endl;

    cout << "After filling list, " << "the elements are:" << endl;

    //Output the elements of list

```

```

    printArray(list, ARRAY_SIZE);
    cout << endl;

    //Find and output the sum of the elements of list
    cout << "The sum of the elements of " << "list is: " <<
sumArray(list, ARRAY_SIZE) << endl;

    //Find and output the position of the largest
//element in list
    cout << "The position of the largest "<< "element in list is:
" << indexLargestElement(list, ARRAY_SIZE)<<endl;

    //Find and output the largest element in list
    cout << "The largest element in "<< "list is: " <<
list[indexLargestElement(list, ARRAY_SIZE)] <<endl;

return 0;

}

//Function Definitions
void fillArray(int arr[],int sizeX)
{
    int index;
    for (index = 0; index < sizeX; index++)
        cin >> arr[index];
}

/////////////////////////////////////////////////////////////////
void printArray(const int arr[],int sizeX)
{
    int index;
    for (index = 0; index < sizeX; index++)
        cout << arr[index] << " ";
}

/////////////////////////////////////////////////////////////////
int sumArray(const int arr[],int sizeX)
{
    int index;
    int sum = 0;
    for (index = 0; index < sizeX; index++)
        sum = sum + arr[index];
    return sum;
}

/////////////////////////////////////////////////////////////////

```

```

int indexLargestElement(const int arr[],int sizeX)
{
    int index;
    int maxIndex = 0; //assume the first element is the largest
    for (index = 1; index < sizeX; index++)
        if (arr[maxIndex] < arr[index])
            maxIndex = index;
    return maxIndex;
}

```

TASKS:

1. Write a C++ program which take 20 integer values as input from user and print the following:
 - a. The count of positive values.
 - b. The count of negative values.
 - c. The count of odd values.
 - d. The count of even values.
 - e. The count of zero values.
 - f. The largest element of array.
 - g. The smallest element of array.
2. Write a C++ program which take 10 integer values as input from user and store them in an array. Take an integer value input from user and check whether the integer value present in array or not. Print on screen that whether integer value found in array or not and display the number of times integer value appears in the array.

Sample Output:

```

Enter Integers: 5 6 7 22 7 3 6 8 2 4
Enter value to find: 7
Integer 7 is present in array.
Integer 7 appears 2 times appear in array.

```

3. Write a C++ program which take 10 integer values as input from user and store them in an array. Copy all the elements in another array but in reverse order.

Steps:

Declare another array of same size. Copy last element of arr1 in first element of arr2, second last of arr1 into second of arr2 and so on...

For array of 5 elements:

```

arr2[0] = arr1[4]
arr2[1] = arr1[3]

```

```
arr2[2] = arr1[2]  
arr2[3] = arr1[1]  
arr2[4] = arr1[0]
```

HINT: there will be 2 counters in the loop. One increasing and other decreasing.

Sample Output:

```
Array 1: 22 5 3 7 34  
Array 2: 34 7 3 5 22  
  
Process returned 0 (0x0)   execution time : 0.078 s  
Press any key to continue.
```

CSC 103 – Programming Fundamentals - Lab

Week 12: 2D Arrays

Statement Purpose:

In this lab, students will know about the basic concepts of two – dimensional Arrays and passing 2D-arrays to the functions.

Activity Outcomes:

This lab teaches you the following topics:

- Create two-dimensional Arrays
- Accessing two-dimensional array components
- Different two-dimensional array's operation
- Passing two-dimensional arrays to functions

Two-dimensional array:

A collection of a fixed number of components arranged in rows and columns (that is, in two dimensions), where in all components are of the same type.

The syntax for declaring a two-dimensional array is:

```
dataType arrayName[intExp1][intExp2];
```

Where in **intExp1** and **intExp2** are constant expressions yielding positive integer values. The two expressions, **intExp1** and **intExp2**, specify the number of rows and the number of columns, respectively, in the array. The statement:

```
double sales[10][5];
```

declares a two-dimensional array **sales** of 10 rows and 5 columns, in which every component is of type **double**.

Accessing Array Components:

To access the components of a two-dimensional array, you need a pair of indices: one for the row position and one for the column position. The statement:

```
sales[5][3] = 25.75;
```

stores 25.75 into row number 5 and column number 3 (that is, the sixth row and the fourth column) of the array **sales**.

Activity 1: (Processing Two-Dimensional Arrays)

A two-dimensional array can be processed in three ways:

1. Process the entire array.
2. Process a particular row of the array, called row processing.
3. Process a particular column of the array, called column processing.

This program demonstrates the basic operations on a two-dimensional array:

- Initializing
- Inputting data
- Outputting data stored in an array
- Sum of each individual row
- Sum of each individual column
- Finding the largest element in each row
- Finding the largest element in each column

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

const int NUMBER_OF_ROWS = 3; //This can be set to any number.
const int NUMBER_OF_COLUMNS = 3; //This can be set to any
number.

using namespace std;

int main()
{
    int matrix[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
    int row,col;
    int sum;
    int largest;

    //Initializing an array
    //The following nested for loop initializes every component of
    //the array matrix to 0.
        for (row = 0; row < NUMBER_OF_ROWS; row++)
            for (col = 0; col < NUMBER_OF_COLUMNS; col++)
                matrix[row][col] = 0;

    //Reading data into an array.
    // The following for loop inputs data into each component of
    //matrix:
        for (row = 0; row < NUMBER_OF_ROWS; row++)
            for (col = 0; col < NUMBER_OF_COLUMNS; col++)
                cin >> matrix[row][col];
```

```

//Printing an array
//The following nested for loops print the components of matrix,
//one row per line:
    for (row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (col = 0; col < NUMBER_OF_COLUMNS; col++)
            cout << setw(5) << matrix[row][col] << " ";
        cout << endl;
    }

//Sum of each individual row
// Following is the C++ code to find the sum of each individual
//row:
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
    sum = 0;
    for (col = 0; col < NUMBER_OF_COLUMNS; col++)
        sum = sum + matrix[row][col];
    cout << "Sum of row " << row + 1 << " = " << sum << endl;
}

//Sum of each individual column
//The following nested for loop finds the sum of each individual
//column:
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{
    sum = 0;
    for (row = 0; row < NUMBER_OF_ROWS; row++)
        sum = sum + matrix[row][col];
    cout << "Sum of column " << col + 1 << " = " << sum << endl;
}

//Largest element in each row
for (row = 0; row < NUMBER_OF_ROWS; row++)
{
    largest = matrix[row][0]; //Assume that the first element
                           //of the row is the largest.
    for (col = 1; col < NUMBER_OF_COLUMNS; col++)
        if (largest < matrix[row][col])
            largest = matrix[row][col];
    cout << "The largest element in row " << row + 1 << " = " <<
largest << endl;
}

//Largest element in each column
for (col = 0; col < NUMBER_OF_COLUMNS; col++)
{

```



```

    largest = matrix[0][col]; //Assume that the first element
                             //of the column is the largest.
    for (row = 1; row < NUMBER_OF_ROWS; row++)
        if (largest < matrix[row][col])
            largest = matrix[row][col];
    cout << "The largest element in column " << col + 1 << " = " <<
largest << endl;
}

return 0;
}

```

Activity 2: (Passing Two-Dimensional Arrays as Parameters to Functions)

Two-dimensional arrays can be passed as parameters to a function, and they are passed by reference. The base address (that is, the address of the first component of the actual parameter) is passed to the formal parameter.

When storing a two-dimensional array in the computer's memory, C++ uses the row order form. That is, the first row is stored first, followed by the second row, followed by the third row, and so on.

In the case of a one-dimensional array, when declaring it as a formal parameter, we usually omit the size of the array. Because C++ stores two-dimensional arrays in row order form, to compute the address of a component correctly, the compiler must know where one row ends and the next row begins. Thus, when declaring a two-dimensional array as a formal parameter, you can omit the size of the first dimension, but not the second; that is, you must specify the number of columns.

This example shows how to write functions for 2D array processing and declare 2D array as a formal parameter.

```

#include <iostream>
#include <iomanip>

using namespace std;

const int ROWS = 6;
const int COLUMNS = 5;

//Function Prototype
void printMatrix(int matrix[][COLUMNS],int ROWS);
void sumRows(int matrix[][COLUMNS],int ROWS);
void largestInRows(int matrix[][COLUMNS],int ROWS);

int main()
{

```

```

int board[ROWS][COLUMNS]= {{23, 5, 6, 15, 18},
                             {4, 16, 24, 67, 10},
                             {12, 54, 23, 76, 11},
                             {1, 12, 34, 22, 8},
                             {81, 54, 32, 67, 33},
                             {12, 34, 76, 78, 9}};

printMatrix(board, ROWS);
cout << endl;
sumRows(board, ROWS);
cout << endl;
largestInRows(board, ROWS);
return 0;
}

//Function Definitions
/////////////////////////////////////////////////////////////////
void printMatrix(int matrix[][COLUMNS], int noOfRows)
{
    int row, col;
    for (row = 0; row < noOfRows; row++)
    {
        for (col = 0; col < COLUMNS; col++)
            cout << setw(5) << matrix[row][col] << " ";
        cout << endl;
    }
}

/////////////////////////////////////////////////////////////////
void sumRows(int matrix[][COLUMNS], int noOfRows)
{
    int row, col;
    int sum;
    //Sum of each individual row
    for (row = 0; row < noOfRows; row++)
    {
        sum = 0;
        for (col = 0; col < COLUMNS; col++)
            sum = sum + matrix[row][col];
        cout << "Sum of row " << (row + 1) << " = " << sum << endl;
    }
}

/////////////////////////////////////////////////////////////////
void largestInRows(int matrix[][COLUMNS],int noOfRows)
{
    int row, col;
    int largest;

```

```

//Largest element in each row
for (row = 0; row < noOfRows; row++)
{
    largest = matrix[row][0]; //Assume that the first element
                               //of the row is the largest.
    for (col = 1; col < COLUMNS; col++)
        if (largest < matrix[row][col])
            largest = matrix[row][col];
    cout << "The largest element of row " << (row + 1) << " = " <<
largest << endl;
}
}

```

TASKS:

1. Write down the definition of the following function:

```

void copy(const int X[][MAX_COL], int Y[][MAX_COL], int
MAX_ROW);

```

This function receives an input array **X** of size **MAX_ROW x MAX_COL**, and copies this array into array **Y** having the same size.

E.g. if **X** is as give below (i.e. **MAX_ROW=3**, **MAX_COL=5**):

5	12	17	9	3
13	4	8	14	1
9	6	3	7	21

Then, after calling this function, **Y** should also be as given below:

5	12	17	9	3
13	4	8	14	1
9	6	3	7	21

2. Write down the definition of the following function:

```

void Subarray2(const int X[][MAX_COL1], int Y[][MAX_COL2], int
MAX_ROW1, int MAX_ROW2);

```

This function receives an input array **X** of size **MAX_ROW1 x MAX_COL1**, and copies an initial subarray of size **MAX_ROW2 x MAX_COL2** from **X** into array **Y** (see example below). Thus, array **Y** has size **MAX_ROW2 x MAX_COL2**.

E.g. if **X** is as give below:

5	12	17	9	3
13	4	8	14	1
9	6	3	7	21

And **MAX_ROW2=2** , **MAX_COL2=2**, then after completing the function, **Y** should be:

5	12
13	4

Note: Call the above functions in main to test on different arrays and verify that these functions are correct.

3. Write a C++ program that compute the sum of two 2D arrays and store the sum in third 2D array. **MAX_ROW=3** , **MAX_COL=3** . The program must take input in both 2D-arrays of 3x3, add them and store the sum in a third 2D-array. The program should print the values store in 3rd 2D-array.

CSC 103 – Programming Fundamentals - Lab

Week 13: Pointers and Arrays

Pointers and Arrays:

Task 1: Practice the following codes to understand the pointers concept. What will the output of the following C++ code?

Practice Code 1:

```
#include <iostream>

using namespace std;

int main()
{
    int m = 10, n = 5;
    int *ptr_m, *ptr_n;
    ptr_m = &m;
    ptr_n = &n;
    cout<< ptr_m << " " << ptr_n << endl;
    *ptr_m = *ptr_m + *ptr_n;
    *ptr_n = *ptr_m - *ptr_n;
    cout << m <<" "<< *ptr_m <<" "<< n <<" "<< *ptr_n;
}
```

Practice Code 2:

```
// Pass-by-value used to cube a variable's value.
#include <iostream>
using namespace std;

int cubeByValue( int ); // prototype

int main()
{
    int number = 5;
    cout << "The original value of number is " << number;
    number = cubeByValue( number ); // pass number by value to
    //cubeByValue

    cout << "\nThe new value of number is " << number << endl;
} // end main
// calculate and return cube of integer argument
```

```
int cubeByValue( int n )
{
return n * n * n; // cube local variable n and return result
} // end function cubeByValue
```

Practice Code 3:

```
// Pass-by-reference with a pointer argument used to cube a
// variable's value.
#include <iostream>
using namespace std;

void cubeByReference( int * ); // prototype

int main()
{
int number = 5;
cout << "The original value of number is " << number;
cubeByReference( &number ); // pass number address to
//cubeByReference

cout << "\nThe new value of number is " << number << endl;
} // end main
// calculate cube of *nPtr; modifies variable number in main
void cubeByReference( int *nPtr )
{
*nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
} // end function cubeByReference
```

Practice Code 4:

```
#include <iostream>

using namespace std;

void swap(int *i, int *j)
{
    int temp = *i;
    *i = *j;
    *j = temp;
}

int main()
{
    int x=10, y=20;
    swap(&x, &y);
    cout << x << " " << y << endl;
```

```
    return 0;
}
```

Practice Code 5:

```
// C++ Program to display address of each element of an array

#include <iostream>
using namespace std;

int main()
{
    double arr[3];
    // declare pointer variable
    double *ptr;

    cout << "Displaying address using arrays: " << endl;

    // use for loop to print addresses of all array elements
    for (int i = 0; i < 3; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }

    // ptr = &arr[0]
    ptr = arr;

    cout<<"\nDisplaying address using pointers: "<< endl;

    // use for loop to print addresses of all array elements
    // using pointer notation
    for (int i = 0; i < 3; ++i)
    {
        cout << "ptr + " << i << " = "<< ptr + i << endl;
    }
    return 0;
}
```

Practice Code 6:

```
// C++ Program to insert and display data entered by using
//pointer notation.

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

```

int main() {
    double arr[5];

    // Insert data using pointer notation
    cout << "Enter 5 numbers: ";
    for (int i = 0; i < 5; ++i) {
        // store input number in arr[i]
        cin >> *(arr + i) ;
    }
    // Display data using pointer notation
    cout << "Displaying data: " << endl;
    for (int i = 0; i < 5; ++i) {
        // display value of arr[i]
        cout << *(arr + i) << endl ;
    }
    return 0;
}

```

Task 2: Write a C++ program to take n integer inputs from the user in an array and calculate the sum of array elements using pointers.

Task 3: Write a C++ program to take n integer values from the user in an array and display array elements in reversing order.

CSC 103 – Programming Fundamentals - Lab

Week 14: Structures

Practice Code 1: (Structures with Functions)

```
#include <iostream>

using namespace std;

// define a structure of person
struct person{

char name[50];
int age;
float salary;
};

// A function which takes a variable of a structure of type person
as an argument
void display(person p){
    cout<<"\n\nDisplay data: (with function)\n\n";
    cout<<"Name:\t"<<p.name;
    cout<<"\nAge:\t"<<p.age;
    cout<<"\nSalary:\t"<<p.salary;
}

person getData(){
    person p;
    cin.ignore(); // this helps to ignore all whitespaces / newline
//from the previous input
    cout<<"\n\nGet Data from the user (with function)";
    cout<<"\nEnter full name:\t ";
    cin.get(p.name,50);
    cout<<"\nEnter age:\t";
    cin>>p.age;
    cout<<"\nEnter salary:\t";
    cin>>p.salary;

    return p; // return a variable p that is a structure of person
//type
}
int main()
{
    // declaring a variable of type person

    person p,p2;
```

```

// get data from the user
cout<<"\nEnter full name:\t ";
cin.get(p.name,50);
cout<<"\nEnter age:\t";
cin>>p.age;
cout<<"\nEnter salary:\t";
cin>>p.salary;

cout<<"\nDisplay data:(without function)\n\n";
cout<<"Name:\t"<<p.name;
cout<<"\nAge:\t"<<p.age;
cout<<"\nSalary:\t"<<p.salary;

    display(p); // function calling. Here the name of the variable
//is passed just like the normal function calling.

    cout<<"\n\nDefine a function which returns structure";
    p2 = getData();
    display(p2);

    return 0;
}

```

TASKS:

Lab Task 1:

- Write down a structure named “**TimeStruct_t**” for saving time information.
 - The structure should contain hours and minutes components along with a component for saving time period **pm/am** information.
- Create a struct variable **time_struct** inside main and initialize its members to values of your choice using struct initializer syntax.
- Display the time in standard time format, e.g. 03:49 pm.
- Update **time_struct** based on user input. Then, display the updated time again.

Lab Task 2:

- Create a function **display_time** with the following prototype, which displays myTime in standard time format.


```
void display_time(TimeStruct_t myTime);
```
- Update the solution of the previous task, so that it calls **display_time** to show time information. The output of the program should be the same as in previous task.
- Also add the function for **getData**.

4. Create a function with the following prototype, which receives **myTime** and updates time by adding 1 minute to **myTime**. (Hint: Don't forget to handle the overflow of hours, minutes and period. E.g. In case of hours overflow, set the hours to 0 and toggle the time period.)

```
TimeStruct_t tic(TimeStruct_t myTime);
```

CSC 103 – Programming Fundamentals - Lab

Week 15: Structures and Functions

Practice Code 1: (Array of structures)

```
#include <iostream>
# define size 10

using namespace std;

struct person
{
    string name;
    int age;
    double salary;
};

void display(person p[], int n)
{
    int i;
    cout<<"\nDisplaying Information\n===== \n";

    for (i=0;i<n;i++){

        cout<<"\nPerson "<<i+1<<endl;
        cout<<"===== "<<endl;
        cout<<"Name:\t"<<p[i].name;
        cout<<"\nAge:\t"<<p[i].age;
        cout<<"\nsalary:\t"<<p[i].salary;
    }
}

void getData(person p[], int n)
{
    int i;

    for (i=0;i<n;i++){

        cout<<"\nPerson "<<i+1<<endl;
        cout<<"===== "<<endl;
        cin.ignore();
        cout<<"Enter name:\t";
        getline(cin,p[i].name);
        cout<<"\nEnter age:\t";
        cin>>p[i].age;
    }
}
```

```

        cout<<"\nEnter salary:\t";
        cin>>p[i].salary;
    }
    display(p,n);
}

int main()
{
    person p[size];
    int n;
    cout<<"\nHow many elements you want to insert in an array:\t";
    cin>>n;
    getData(p,n);
    return 0;
}

```

TASK:

Define a **struct**, **menuItemType**, with two components: **menuItem** of type **string** and **menuPrice** of type **double**.

Write a **C++** program to help a local restaurant automate its breakfast billing system. The program should do the following:

- Show the customer the different breakfast items offered by the restaurant.
- Allow the customer to select more than one item from the menu.
- Calculate and print the bill.

Assume that the restaurant offers the following breakfast items (the price of each item is shown to the right of the item):

Plain Egg	\$1.45
Bacon and Egg	\$2.45
Muffin	\$0.99
French Toast	\$1.99
Fruit Basket	\$2.49
Cereal	\$0.69
Coffee	\$0.50
Tea	\$0.75

Use an array, **menuList**, of the **struct menuItemType**. Your program must contain at least the following functions:

- a) Function **getData**: This function loads the data into the array **menuList**.
- b) Function **showMenu**: This function shows the different items offered by the restaurant and tells the user how to select the items.

- c) Function **printCheck**: This function calculates and prints the check. (Note that the billing amount should include a 5% tax.)

A sample output is:

```
Welcome to Johnny's Restaurant
Bacon and Egg      $2.45
Muffin             $0.99
Coffee             $0.50
Tax                $0.20
Amount Due         $4.14
```

CSC 103 – Programming Fundamentals - Lab

Week 16: File Handling

Task 1:

Practice the file handling codes discuss in theory lecture.

Task 2:

Open a "**file1.txt**" file for writing (use **ofstream**). Read three variables **x (integer)**, **y (double)** and **z (integer)** from the user (keyboard) and write them to the file in the same order. For writing the above variables to file, use the **ofstream** variable with **<<** (stream extraction operator) just like **cout**.

Note that the file opening operation should be first validated before writing data to the file. Also, close the file before ending the program.

Task 3:

Open the "**file1.txt**" file for reading (use **ifstream**) and read the three variables written to file in the previous problem. Note that the variables should be read in the same order in which they were written. For reading the above variables from the file, use the **ifstream** variable with **>>** (stream insertion operator) just like **cin**.

After reading the three variables, your program should display a welcome message like **Sum of Integers: (200 + 300) = 500. Double: 5.314** (Note that 200 and 300 were integers read from the file and 500 is their sum performed by the program.)

Task 4:

What is the output of the following codes:

Practice Code 1:

```
//The purpose of code to find the sum of the numbers in each line
//and output the sum. Moreover, assume that this data is to be
//read from a file numbers.txt.
```

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    int counter;
    int num, sum;
```

```

    ifstream infile;
    infile.open("numbers.txt",ios::in);
    counter = 0; //Line 1
    infile >> num; //Line 2
    while (infile) //Line 3
    { //Line 4
        sum = 0; //Line 5
        while (num != -999) //Line 6
        { //Line 7
            sum = sum + num; //Line 8
            infile >> num; //Line 9
        } //Line 10
        cout << "Line " << counter + 1 << ": Sum = " << sum <<
endl; //Line 11
        counter++; //Line 12
        infile >> num; //Line 13
    }

    return 0;
}

```

Practice Code 2:

The data of certain candidates has been seeking the student council's presidential seat.

For each candidate, the data is in the following form:

ID

Name

Votes

The objective is to find the total number of votes received by the candidate. Assume that the data is input from the file ID.txt.

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int ID;
```

```
    char ch;
```

```
    string name;
```

```
    int num, sum;
```

```
    ifstream infile;
```

```
    infile.open("ID.txt");
```

```
    infile >> ID;
```

```
    while (infile)
```

```
    {
```



```

        infile.get(ch);
        getline(infile, name);
        sum = 0;
        infile >> num; // read the first number
        while (num != -999)
        {
            sum = sum + num;
            infile >> num; // read the next number
        }
        cout << "Name: " << name << ", Votes: " << sum
<< endl;
        infile >> ID; // begin processing the next line
    }

    return 0;
}

```