

MANIPAL INSTITUTE OF TECHNOLOGY

Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.



CERTIFICATE

This is to certify that Ms./Mr.

Reg. No. Section: Roll No: has
satisfactorily completed the lab exercises prescribed for Problem Solving Using
Computers Lab [CSE 1011] of First Year B. Tech. Degree at MIT, Manipal, in the
academic year 2014-2015.

Date:

Signature
Faculty in Charge

Signature
Head of the Department

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	COURSE OBJECTIVES AND OUTCOMES	i	
	EVALUATION PLAN	i	
	INSTRUCTIONS TO THE STUDENTS	ii – iii	
	SAMPLE LAB OBSERVATION NOTE PREPARATION	iv – v	
1	INTRODUCTION TO COMPUTERS AND PROGRAMMING IDE	1 –21	
2	SIMPLE C++ PROGRAMS	22 – 24	
3	CONTROL STRUCTURES - DECISION MAKING AND BRANCHING	25 – 33	
4	CONTROL STRUCTURES - LOOPING	34 – 39	
5	1D ARRAYS	40 – 45	
6	2D ARRAYS	46 – 52	
7	STRINGS	53 – 58	
8	FUNCTIONS	59 – 66	
9	RECURSIVE FUNCTIONS	67 – 71	
10	MATLAB – TUTORIAL 1	72 – 85	
11	MATLAB – TUTORIAL 2	86 – 92	
12	MODELING WITH SIMULINK	93 – 99	
	REFERENCES	100	
	C++ QUICK REFERENCE SHEET		

Course Objectives

- To develop the programming skills using basics of C++ language
- To write algorithms and draw flowchart for various problems
- To write, compile and debug programs in C++ language
- Provide skills to code algorithms for Numerical Methods
- To demonstrate basics of MATLAB programming

Course Outcomes

At the end of this course, students will be able to

- Develop, execute and document C++ programs.
- Write programs that implement Numerical Methods
- Demonstrate the programming skills in MATLAB and SIMULINK

Evaluation plan

- Internal Assessment Marks : 60%
 - ✓ Continuous evaluation component (for each experiment):10 marks
 - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce
 - ✓ Total marks of the 12 experiments reduced to marks out of 60
- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must Sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the Lab record
- Prescribed textbooks and class notes can be kept ready for reference if required
- Write the algorithm in the Lab Manual (required only till week 3)
- Draw the corresponding flowchart in the Lab Manual
 - Flowcharts need not be drawn for the programs from week 4 (Arrays) onwards

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
 - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
 - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
 - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
 - Statements within the program should be properly indented.
 - Use meaningful names for variables and functions.
 - Make use of constants and type definitions wherever needed.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.

- The exercises for each week are divided under three sets:
 - Solved exercise
 - Lab exercises - to be completed during lab hours
 - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed during the repetition class with the permission of the faculty concerned but credit will be given only to one day's experiment(s).
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

Sample lab observation note preparation

LAB NO: 1

Date:

Title: SIMPLE C++ PROGRAMS

1. Program to find area of the circle. (Hint: $\text{Area} = 3.14 * r * r$)

Aim: To write an algorithm, draw flow chart and program in C++ to find area of a circle and verify the same with various inputs(radius).

Algorithm:

Name of the algorithm: Compute the area of a circle

Step1: Input radius

Step 2: [Compute the area]

$\text{Area} \leftarrow 3.1416 * \text{radius} * \text{radius}$

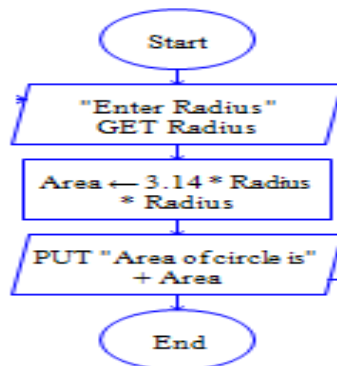
Step 3: [Print the Area]

Print 'Area of a circle =', Area

Step 4: [End of algorithm]

Stop

Flow Chart:



Program:

```
//student name_lab1_1.cpp

//program to find area of circle

#include<iostream.h>

#include<conio.h>

void main()

{

    int radius;

    float area;

    cout<<"Enter the radius"<<endl;

    cin>>radius;


    area=3.14*radius*radius;

    cout<<"The area of circle for given radius is:"<<area;

    getch();

}
```

Sample input and output:



```
C:\TC\BIN>tc.exe
Enter the radius
12
The area of circle for given radius is:452.160004_
```


LAB NO: 1

Date:

INTRODUCTION TO COMPUTERS AND PROGRAMMING IDE**Objectives:**

In this lab, student will be able to:

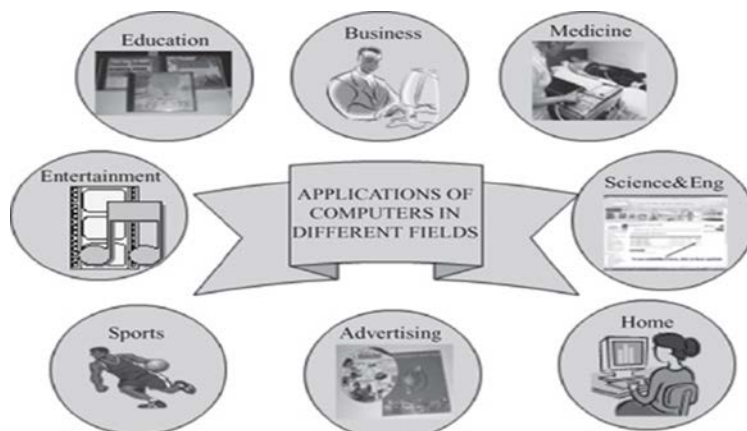
1. Introduction to Computers
2. Core functionality of the computer system.
3. Introduction to problem solving & programing paradigms
4. Algorithms and Flowcharts
5. Understand different components of a C++ program.
6. Write, compile and execute simple C++ programs.

I. INTRODUCTION TO COMPUTER HARDWARE AND SYSTEM SOFTWARE CONCEPTS**Introduction to computer**

- ✓ What is a computer?
 - ✓ an electronic device
 - ✓ operates under the control of instructions stored in its own memory unit
 - ✓ accepts data (Input)
 - ✓ manipulates data (Process)
 - ✓ produces information (Output)

The key characteristics of a computer are

- ✓ Speed
- ✓ Accuracy
- ✓ Diligence – does not get tired
- ✓ Storage capability
- ✓ Versatility – doc preparation, play a music at the same time

Applications of a Computer

Various Computing devices

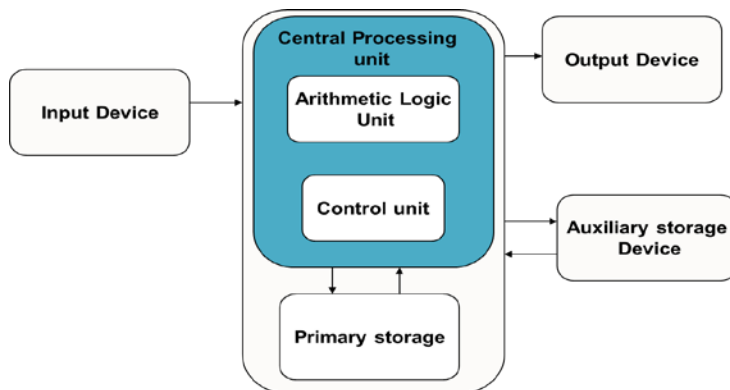


What Does A Computer Do?

Computers can perform **four** general operations, which comprise the information processing cycle.

- Input
- Process/Computation
- Storage
- Output

Block Diagram of a Computer



Input devices

- An external device connected to the CPU
- To feed data and instructions for solving the problem
 - Keyboard
 - Mouse
 - Joystick
 - Light pen



- Trackball
- Optical Scanner
- Voice input

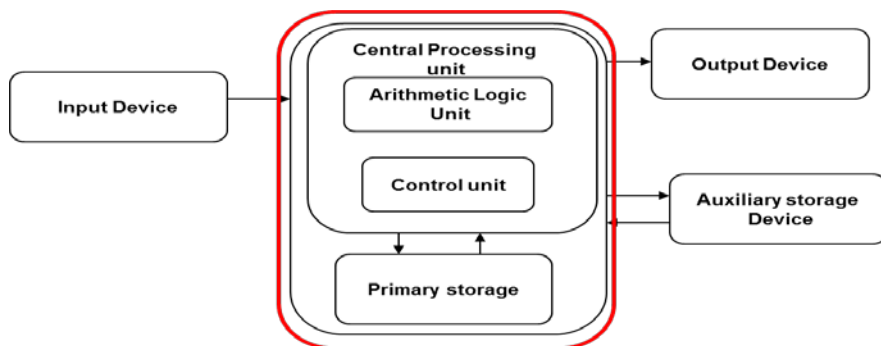
Output devices

- To display the results
 - Printer
 - Plotter
 - Plasma display panels
 - LCD displays
 - Voice output



System unit

- ✓ The Central Processing Unit + Memory Unit
- ✓ Data and instructions received from the input device are stored and processed in this unit



Central Processing Unit

- Data and instructions are processed in CPU
- Consists of two functional units
 - Control Unit (CU)
 - Arithmetic and Logic Unit (ALU)
- Central Processing Unit (CPU) or the processor is also often called the *brain of computer*. In addition, CPU also has a set of registers which are temporary storage areas for holding data, and instructions.

Arithmetic and Logical unit

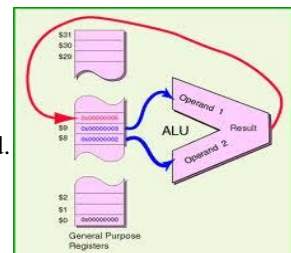
- Performs arithmetic and logical operations:
- Example: arithmetic(+, -, *, / etc..) and logical (AND, OR, NOT, <, = etc..) operations

Control unit

- Controls the order in which your program instructions are executed.

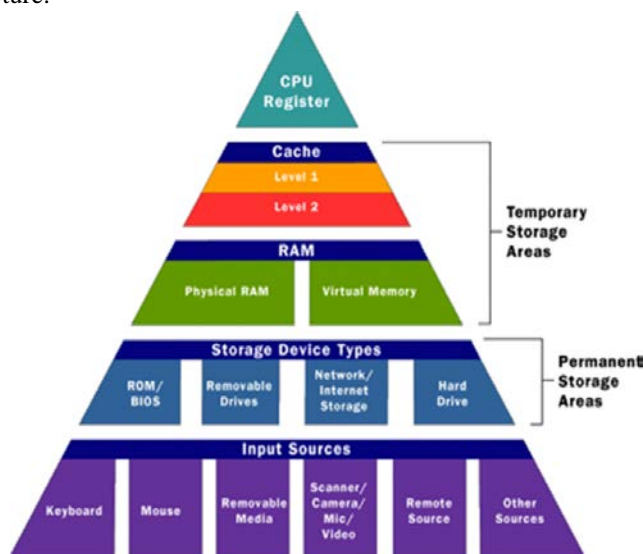
Functions of CU:

- Fetches data and instructions to main memory
- Interprets these instructions
- Controls the transfer of data and instructions to and from main memory
- Controls input and output devices.
- Overall supervision of computer system



Memory unit

- Storage device where the data and instructions fed by the user are stored
- An ordered sequence of storage cells, each capable of holding a piece of information
- Each cell has its own unique address
- The information held can be input data, computed values, or your program instructions.
- The computer memory is measured in terms of **bits**, **bytes** and **words**.
- A **bit** is a **binary digit** either 0 or 1.
- A **byte** is unit of memory and is defined as sequence of 8 bits.
- The **word** can be defined as a sequence of 16/32/64 bits or 2/4/8 bytes respectively depending on the machine architecture.



Computer memory classifications

- Main memory-Primary storage
- Secondary memory-Auxiliary storage
- Cache Memory

Main – Primary memory

- Memory where the data and instructions, currently being executed are stored
- Located outside CPU
- High speed
- Data and instructions stored get erased when the power goes off
- Also referred as **primary** / temporary memory
- Semiconductor memory
- Measured in terms of megabytes and gigabytes

Secondary storage devices

- Main memory is volatile and limited
- Hence it is essential for other types of storage devices where programs and data can be stored when they are no longer being processed
- Installed within the computer box at the factory or added later as needed Non-volatile memory.
- Made up of magnetic material

- Stores large amount of information for long time
- Low speed
- Holds programs not currently being executed

Primary storage: RAM & ROM

RAM stands for Random Access Memory

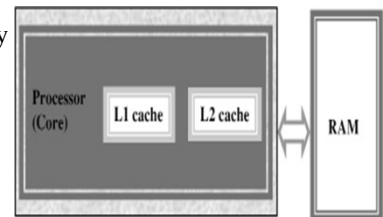
- Read and write memory
- Information typed by the user are stored in this memory
- Any memory location can be accessed directly without scanning it sequentially (random access memory)
- During power failure the information stored in it will be erased → volatile memory

ROM stands for Read Only Memory

- Permanent memory and non-volatile
- Contents in locations in ROM cannot be changed
- Stores mainly stored program and basic input output system programs

Cache memory

- High speed memory placed between CPU and main memory
- Stores data and instructions currently to be executed
- More costlier but less capacity than main memory
- Users can not access this memory

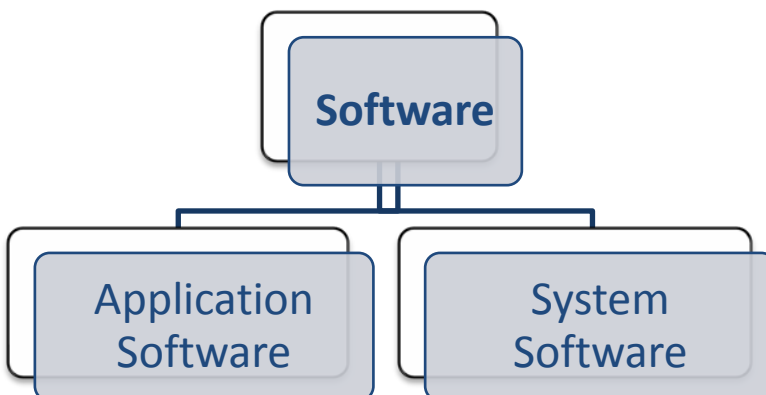


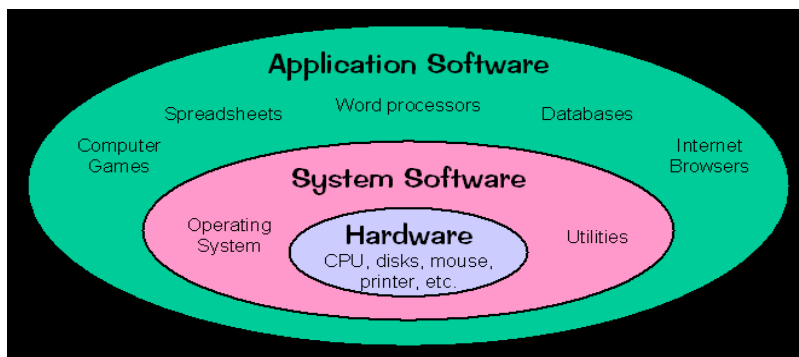
II. INTRODUCTION TO SOFTWARE

INSTRUCTION TO COMPUTER

- **Program or software** - a detailed list of instructions that tells the computer what exactly to do.

Classifications of software





SYSTEM SOFTWARE

- System software consists of programs that manage the computer resources.

Divided into three classes

1. Operating System (OS). eg: Windows, Linux, DOS

- OS is an integrated collection of programs which make the computer operational and help in executing user programs.
- It acts as an interface between the man and machine.
- It manages the system resources like memory, processors, input-output devices and files.

2. System support software. eg: disk format programs

3. System Development software. eg: language translators and debugging tools

APPLICATION SOFTWARE

- Application Software is directly responsible for helping users solve their problems
- Example

- Word processing
- Electronic spreadsheet
- Database
- Presentation graphics



COMPUTER LANGUAGES

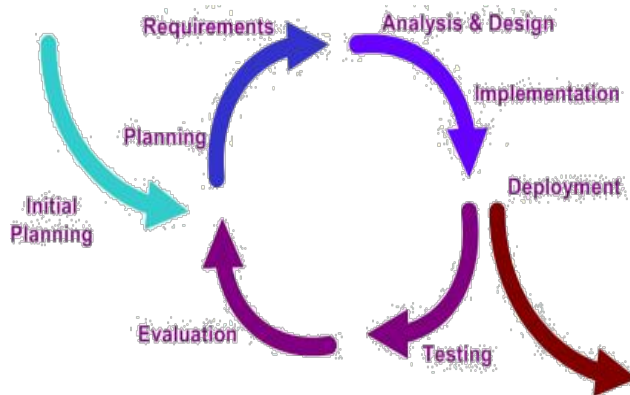
- **Machine Language**- The only programming language available in earlier days
 - Consists of only 0's and 1's; e.g.:- 10101011
- **Symbolic language or Assembly language**-
 - symbols or mnemonics are used to represent instructions
 - hardware specific
 - e.g. ADD X,Y; Add the contents of y to x
- **High-level languages**-English like language using which the programmer can write programs to solve a problem.
 - more concerned with the problem specification
 - not oriented towards the details of computer

- e.g.: C, C++, C#, FORTRAN, BASIC, Pascal etc.

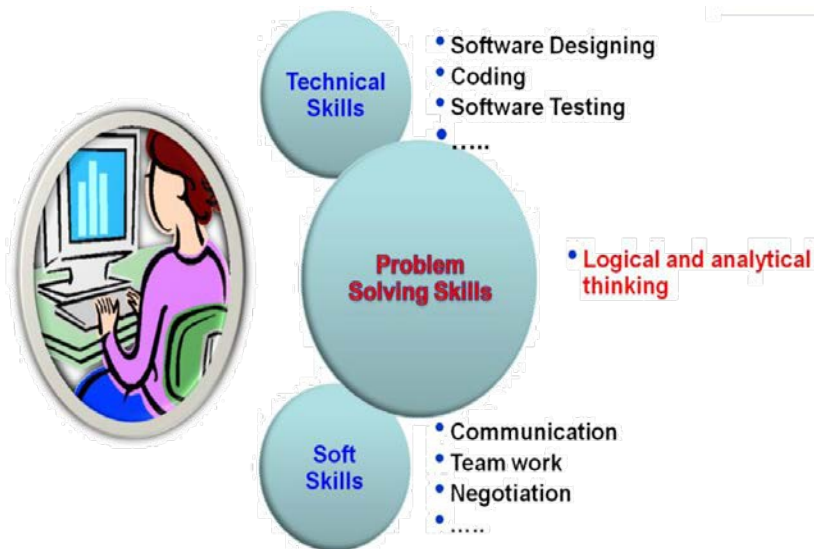
Language Translators

- Compiler: Program that translates entire high level language program into machine language at a time.
e.g.: C, C++ compilers.
- Interpreter: Program which translates one statement of a high level language program into machine language at a time and executes it.
e.g.: Basic Interpreters, Java Interpreters.
- Assembler: Program which translates an assembly language program into machine language.
e.g.: TASM(Turbo ASseMbler), MASM(Macro ASseMbler).

III. INTRODUCTION TO PROBLEM SOLVING



Skill set required for SW Engineers Problem



Definition: A problem is a puzzle that requires logical thought or mathematics to solve.

A puzzle could be a set of questions on a scenario which consists of **description of reality** and set of **constraints** about the scenario.

Scenario: Infosys Mysore campus has a library. The librarian issues books only to Infosys employees.

Careful observation suggests...

Description of reality : There is a library in Infosys Mysore campus and there is a librarian in the library.

Constraint : Librarian issues books only to Infosys employees

Questions about the scenario:

1. How many books are there in the library?
2. How many books can be issued to an employee?
3. Does the librarian issue a book to himself?
- etc...

LOGIC

Definition : A method of human thought that involves thinking in a linear, step by step manner about how a problem can be solved

Logic is a language for reasoning. It is a collection of rules we use when doing reasoning.

Example

John's mum has four children.

- The first child is called April.
- The second May.
- The third June.

What is the name of the fourth child?



IMPORTANCE OF LOGIC IN PROBLEM SOLVING

Solution for any problem (summation of two numbers) requires three things

Input	: Input values (e.g. 2 and 3)
Process	: Process of Summation
Output	: Output after process (e.g. sum of numbers (5))

The process part(e.g. Summation) of the solution requires logic (e.g. How to sum)

Or in other words based on the logic the process is developed

For solving a problem there may be multiple valid logics some may be simple and others may be complex

For example: Determine whether a given number is prime or not?

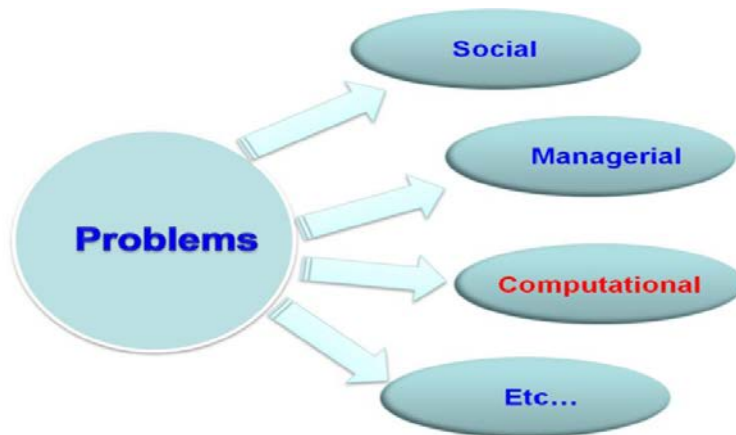
Logic 1: Divide the number by all the numbers from 2 to one less than the number and if for all the division operations, the remainder is non zero, the number is prime otherwise it is not prime

Logic 2: Same as logic1 but divide the number only from 2 to the number /2

Logic 3: Same as Logic 1 but divide the number only from 2 to square root of the number

As a software engineer our job is to find out appropriate logic to solve given problem

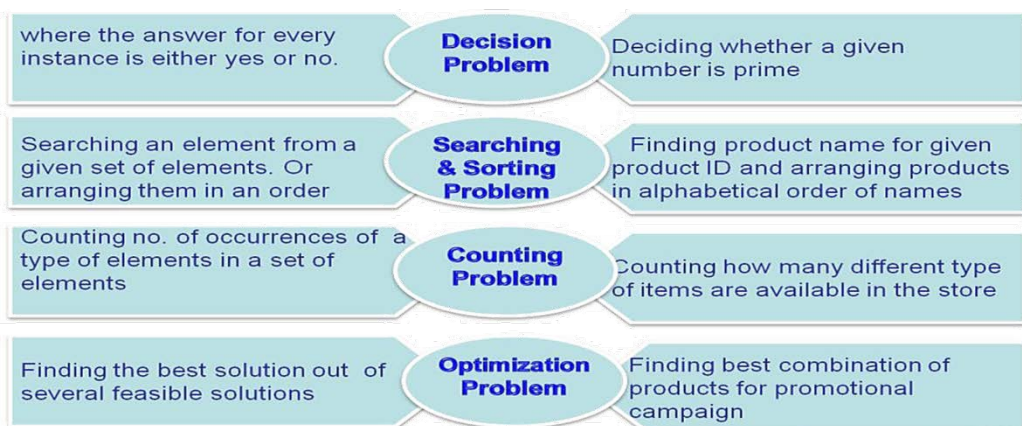
TYPES OF PROBLEMS



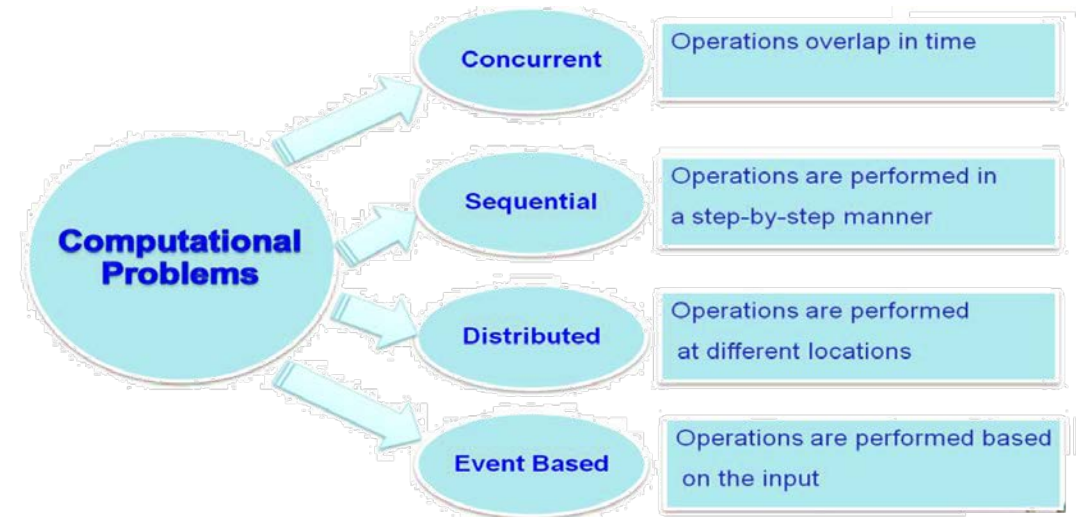
COMPUTATIONAL PROBLEMS

Computation Definition: Computation is the process of evolution from one state to another in accordance with some rules.

Broad applications of Computational Problem

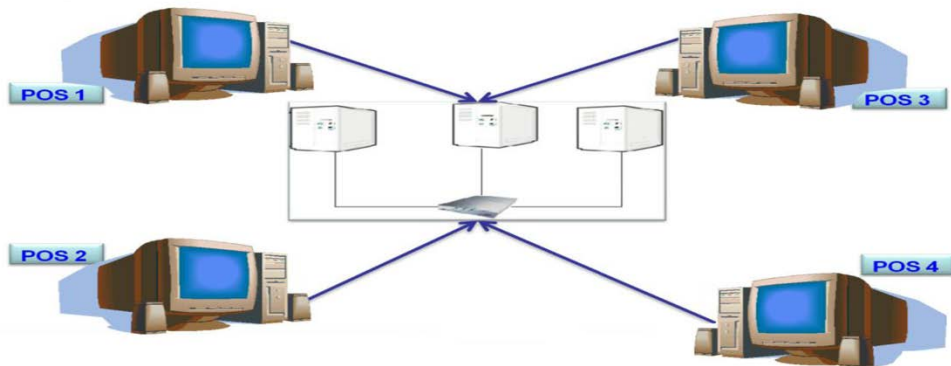


CLASSIFICATION OF COMPUTATIONAL PROBLEMS



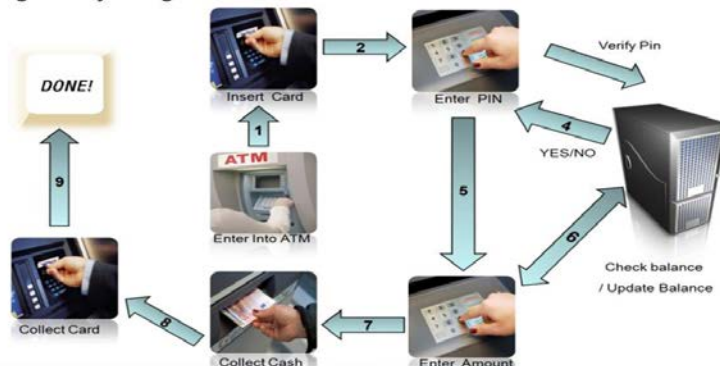
DISTRIBUTED / CONCURRENT COMPUTATION

Billing process at multiple Point of Sales of a retail shop



SEQUENTIAL / EVENT BASED COMPUTATION

Withdrawing money using ATM



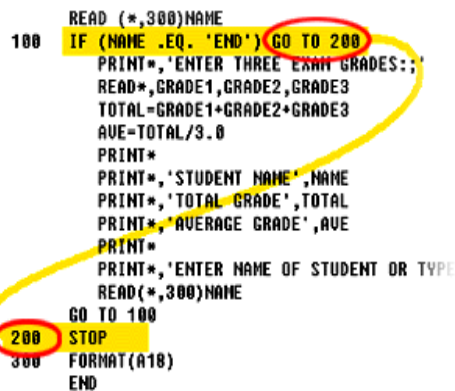
IV. PROGRAMMING PARADIGMS AND METHODOLOGIES

- The programming methodologies and Paradigms may be put to use during the design and the build phases of the SDLC.
- Programming Approaches
 - **Unstructured Programming**
 - **Structured Programming [C]**
 - **Object Oriented Programming [OOP; C++]**

Unstructured Programming

- Entire program code is written in a single continuous **main** program
- The main program operates on the data directly

Ex: Basic Language



```

      READ (*,300)NAME
100  IF (NAME .EQ. 'END') GO TO 200
      PRINT*, 'ENTER THREE EXAM GRADES:;'
      READ*, GRADE1, GRADE2, GRADE3
      TOTAL=GRADE1+GRADE2+GRADE3
      AVE=TOTAL/3.0
      PRINT*
      PRINT*, 'STUDENT NAME', NAME
      PRINT*, 'TOTAL GRADE', TOTAL
      PRINT*, 'AVERAGE GRADE', AVE
      PRINT*
      PRINT*, 'ENTER NAME OF STUDENT OR TYPE
      READ(*,300)NAME
      GO TO 100
200  STOP
300  FORMAT(A10)
      END
  
```

UNSTRUCTURED PROGRAMMING LIMITATIONS

As the size of the program increases

- ✓ Difficult to understand and modify the program.
- ✓ If something needs to be done more than once it must be re-typed and it is difficult to test a particular part of the code.
- ✓ All parts of the program can access the data (whether required or not). Hence error in one part of the program can affect the other parts of the program also, there exist more chances of accidental modification to the data

STRUCTURED PROGRAMMING

- ✓ Structured or Procedural programming attempts to divide the problem into smaller subtasks or modules which interact with other to complete the given task thereby making the complexity of code manageable.
- ✓ Modular programming is a software design technique that divides the tasks into sub tasks such that each sub task can be developed independently.
- ✓ The aim of structured programming is to clearly define the structure of the program before writing program code. Ex: ALGOL, Pascal, Ada etc.

STRUCTURED PROGRAMMING LIMITATIONS

- ✓ Structured programming focuses mainly on the algorithm and not on the data on which these algorithms operate
- ✓ Structured programming is a good model for small-medium sized projects

OBJECT ORIENTED PROGRAMMING

- ✓ Object oriented programming model is used when the complexity of the problem is more.
- ✓ Object Oriented Programming (OOP) is often natural and powerful way of representing a real-life situation or problem.
- ✓ An Object has two components
 1. State information which describes the current characteristics of the objects and
 2. Behavior which describes how it interacts with other objects

Ex: C++, JAVA languages

V. INTRODUCTION TO ALGORITHMS

To learn and appreciate the following concepts

- ✓ Introduction to algorithms
- ✓ Algorithms for simple problems

At the end of session the student will be able to write

- ✓ Algorithms for simple problems

Algorithm

- ✓ A *step by step procedure* to solve a particular problem
- ✓ Named after Arabic Mathematician Abu Jafar Mohammed Ibn Musa Al Khowarizmi

Algorithmic Notations

- **Start**[Begin of algorithm]
- **Name of the algorithm**[mandatory]
[gives a meaningful name to the algorithm based on the problem]
- **Step Number**[mandatory]
[indicate each individual simple task]
- **Explanatory comment**[optional]
[gives an explanation for each step, if needed]
- **Termination**[mandatory]
[tells the end of algorithm]

Example - Compute the area of circle

Name of the algorithm: Compute the area of a circle

Step 1: Start

Step 2: Input radius

Step 2: [Compute the area]

$\text{Area} \leftarrow 3.1416 * \text{radius} * \text{radius}$

Step 3: [Print the Area]

Print 'Area of a circle =', Area

Step 4: [End of algorithm]

Stop

Example - Largest of 3 Numbers

Name of the algorithm: Find largest of 3 numbers

Step 1: Start

Step 2: [Read the values of A, B and C]

Read A, B, C

Step 3: [Compare A and B]

IF $A > B$ go to step 4

Step 4: [Otherwise compare B with C]

IF $B > C$ then

Print 'B' is largest'

Else

Print 'C' is largest'

Go to Step 6

Step 5: [Compare A and C for largest]

IF $A > C$ then

Print 'A' is largest'

Else

Print 'C' is largest'

Step 6: [End of the algorithm]

Stop

Example - Factorial of a number

Name of the algorithm: Compute the factorial of a number

Step1: start

Step 2: Input N

Step 3: $\text{fact} \leftarrow 1$

Step 4: For count=1 to N in step of 1 do

begin

$\text{fact} \leftarrow \text{fact} * \text{count}$

end

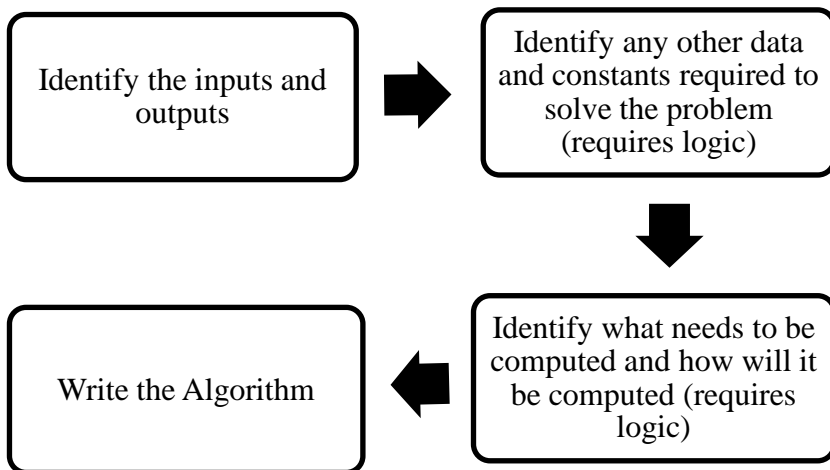
Step 5: Print 'fact of N=', fact

Step 6: [End of algorithm]

Stop

Properties of an algorithm

- ✓ **Finiteness** – Must terminate after a finite numbers of steps.
- ✓ **Definiteness** – Action in each step to be carried out must be rigorously and unambiguously specified.
- ✓ **Input** – Has zero or more inputs that are provided to it initially or dynamically.
- ✓ **Output** – Has one or more outputs; quantities that have a specified relation to inputs.
- ✓ **Effectiveness** – Operations must all be sufficiently basic.

Steps to develop an algorithm

VI. INTRODUCTION TO FLOWCHARTS

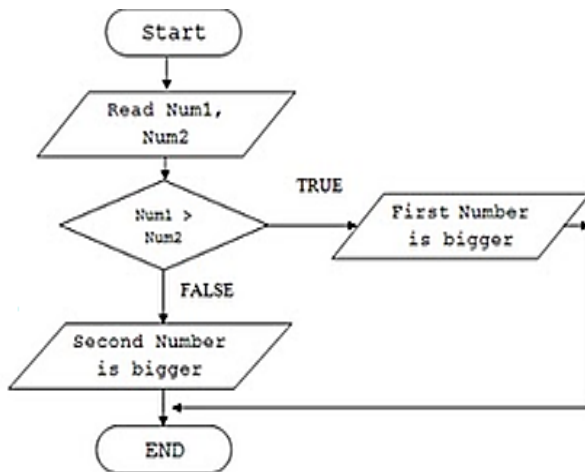
Representation of computation

- ✓ In Computer Science, **Flow chart** is used to represent algorithm which basically provides a solution to any computational problem.

- **Flowchart:** Agraphical/pictorial representation of computation

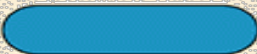

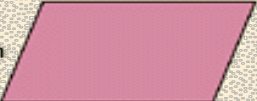
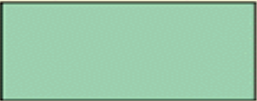
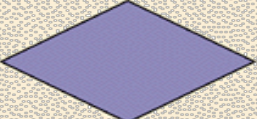
-

Algorithm and Flow chart



Step 1: start
 Step 2: input num1,num2
 Step 3: if num1>num2 then
 print num1 is bigger
 otherwise
 print num2 is bigger
 Step 4: end

Basic Flowchart Symbols

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g. PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).

Algorithm and Flowchart for area of the circle**FLOWCHART**

Name of the algorithm: Compute the area of a circle

Step1: Input radius

Step 2: [Compute the area]

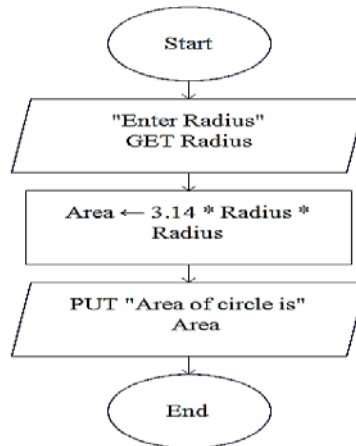
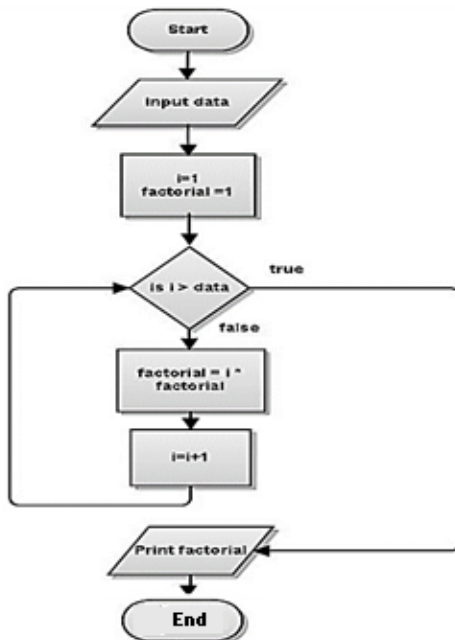
$\text{Area} \leftarrow 3.1416 * \text{radius} * \text{radius}$

Step 3: [Print the Area]

Print 'Area of a circle =', Area

Step 4: [End of algorithm]

Stop

**Flowchart to find factorial of given number**

Name of the algorithm:

Compute the factorial of a number

Step1: start

Step 2: Input data

Step 3: factorial $\leftarrow 1$

Step 4: For i=1 to N in step of 1 do
begin
factorial \leftarrow factorial * i
end

Step 5: Print 'fact of N=', factorial

Step 6: [End of algorithm]

Stop

Key features of flow charts

- ✓ Diagrammatic / visual / graphical representation of computation of an algorithm/pseudo code
- ✓ Easier to understand and analyze the problem and it's solution before programming
- ✓ Machine independent
- ✓ Well suited for any type of logic

VII. TURBO C ++ EDITOR

- ✓ Menu Bar
- ✓ Key Short cuts (F1, F2, ALT-F9, CTRL-F9, ALT-F5)
- ✓ Context Help (CTRL-F1)
- ✓ Tracing an Program Execution- (CTRL-F7, F7, F8, F4)

Introduction

This lab course provides an introduction to computer programming using the C++ language. In this lab, the student will be able to write, see and debug their first program.

Running a sample C++ program

Let's look at C++ program implementation in steps by writing, storing, compiling and executing a sample program

- Create a directory with section followed by roll number (to be unique); e.g. A21.
- As per the instructions given by the lab teacher, create *InchToCm.cpp* program.
 - Open a new notepad file and type the given program
- Save the file with name and extension as “*InchToCm.cpp*” into the respective directory created.

Sample Program (*InchToCm.cpp*):

```
// InchToCm.cpp
// this program inputs a real number inches and outputs its centimeter equivalent
// (also a real number)
#include <iostream.h>
#include<conio.h>
void main() // int main()
{
    float centimeters, inches;
    cout<< "This program converts inches to centimeters" <<endl;
    cout<< "Enter a number> ";
    cin>> inches;
    centimeters = inches * 2.54;
    cout<< inches << " inches is equivalent to " << centimeters
        << " centimeters" <<endl;
    getch(); // return 0;
} // end main
```

- Run the program as per the instructions given by the lab teacher.
 - Compile the saved program and run it either by using keyboard short cuts (CTRL+F9) or through the menu.

PROGRAM STRUCTURE AND PARTS

Comments

The first line of the file is:

```
// InchToCm
```

This line is a comment. Let's add a comment above the name of the program that contains the student's name.

- Edit the file. Add the student's name on the top line so that the first two lines of the file now look like:

```
// student's name
```

```
// InchToCm
```

Comments tell people reading the program something about the program. The compiler ignores these lines.

Preprocessor Directives

After the initial comments, the student should be able to see the following line:

```
#include <iostream.h>
```

This is called a preprocessor directive. It tells the compiler to do something. Preprocessor directives always start with a # sign. In this case, the preprocessor directive includes the information in the file in stream as part of the program. Most of the programs will almost always have at least one include file. These header files are stored in a library that shall be learnt more in the subsequent labs.

The function main()

The next non-blank line *void main ()* gives the name of a function. There must be exactly one function named *main* in each C++ program, and *main* is the function where program execution starts when the program begins running. The *void* before *main ()* indicates the function is not returning any value and also to indicate empty argument list to the function. Essentially functions are units of C++ code that do a particular task. Large programs will have many functions just as large organizations have many functions. Small programs, like smaller organizations, have fewer functions. The parentheses following the words *void main* contains a list of arguments to the function. In the case of this function, there are no *arguments*. Arguments to functions tell the function what objects to use in performing its task. The curly braces ({) on the next line and on the last line (}) of the program determine the beginning and ending of the function.

Variable Declarations

The line after the opening curly brace, *float centimeters, inches;* is called a variable declaration. This line tells the compiler to reserve two places in memory with adequate size for a real number (the *float* keyword indicates the variable as a real number). The memory locations will have the names *inches* and *centimeters* associated with them. The programs often have many different variables of many different types.

EXECUTABLE STATEMENTS

Output and Input

The statements following the variable declaration up to the closing curly brace are executable statements. The executable statements are statements that will be executed when the program run. *cout*, the output stream operator (<<) tells the compiler to generate instructions that will display information on the screen when the program run, and *cin*, the input stream operator (>>) reads information from the keyboard when the program run.

Assignment Statements

The statement *centimeters = inches * 2.54;* is an assignment statement. It calculates what is on the right hand side of the equation (in this case *inches * 2.54*) and stores it in the memory location that has the name specified on the left hand side of the equation (in this case, *centimeters*). So *centimeters = inches * 2.54* takes whatever was read into the memory location *inches*, multiplies it by 2.54, and stores the result in *centimeters*. The next statement outputs the result of the calculation.

Return Statement

The last statement of this program, *return 0;* returns the program control back to the operating system. The value 0 indicates that the program ended normally. The last line of every main function written should be *return 0;* this is indicated alternatively to *void main ()*.

Syntax

Syntax is the way that a language must be phrased in order for it to be understandable. The general form of a C++ program is given below:

```
// program name
// other comments like what program does and student's name
#include <appropriate files>
void main()
{
    Variable declarations;
    Executable statements;
} // end main
```

Lab exercises

Most labs will have a synthesis section in which the student will write a program based on what the student has learned. The student is now directed to exercise the following simple programs using C++ code.

- Familiarization of Turbo C++ editor
- Menu Bar
- Key Short cuts (F1, F2, ALT-F9, CTRL-F9, ALT-F5)
- Context Help (CTRL-F1)
- Tracing a ProgramExecution- (CTRL-F7, F7, F8, F4)

Lab exercise

1. Type the following program in C++ Editor and execute it. Mention the errors in lab observation note

```
void main( )
{
    cout<<" This is my first program in C++ ";
}
```

2. Add the following line at the beginning of the above program. Recompile the program. What is the output?

```
#include<iostream.h>
```

3. Make the following changes to the program and observe the errors.

- i. Write Void instead of void
- ii. Write void main (void);
- iii. Remove the semi colon ';'.
- iv. Erase any one of brace '{' or '}'

4. Write down C++ statements to perform the following operations:

i) $z = \frac{4.2(x+y)5/z - 0.52x/(y+z)}{(x+y)^2}$

ii) $x = a^2 + 2ab + b^2$

5. What will be the output of the mix mode use of integers and float:

```
a=5/9;  
b=5.0/9;  
cout<<a<<b;
```

[OBSERVATION SPACE – LAB1]

[OBSERVATION SPACE – LAB1]

LAB NO: 2**Date:****SIMPLE C++ PROGRAMS****Objectives:**

In this lab, student will be able to:

1. Write C++ programs.
2. Compile and execute C++ programs.
3. Debug and trace the programs.

Lab exercises**Simple C++ programs**

Write C++ programs to do the following:

1. Input P, N and R and Compute simple and compound interest.
[Hint: $SI = PNR/100$, $CI = P(1+R/100)^T - P$]
2. Input radius and find the volume and surface area of a sphere.
[Hint: volume = $(4\pi r^3)/3$, Area = $4\pi r^2$]
3. Convert the given temperature in Fahrenheit to Centigrade. [Hint: $C = 5/9(F-32)$]
4. Find the sum, difference, product and quotient of 2 numbers.

Additional exercises

1. Convert the time in seconds to hours, minutes and seconds. [Hint: 1 hr = 3600 sec]
2. Determine how much money (in rupees) is in a piggy bank that contains denominations of 20, 10 and 5 rupee along with 50 paisa coins. Use the following values to test the program: 13 twenty rupee notes, 11 ten rupee notes, 7 five rupee coins and 13 fifty paisa coins. [Hint: answer is Rs.411.5]

[OBSERVATION SPACE – LAB 2]

[OBSERVATION SPACE – LAB2]

[OBSERVATION SPACE – LAB 2]

LAB NO: 3**Date:****CONTROL STRUCTURES-DECISION MAKING AND BRANCHING****Objectives:**

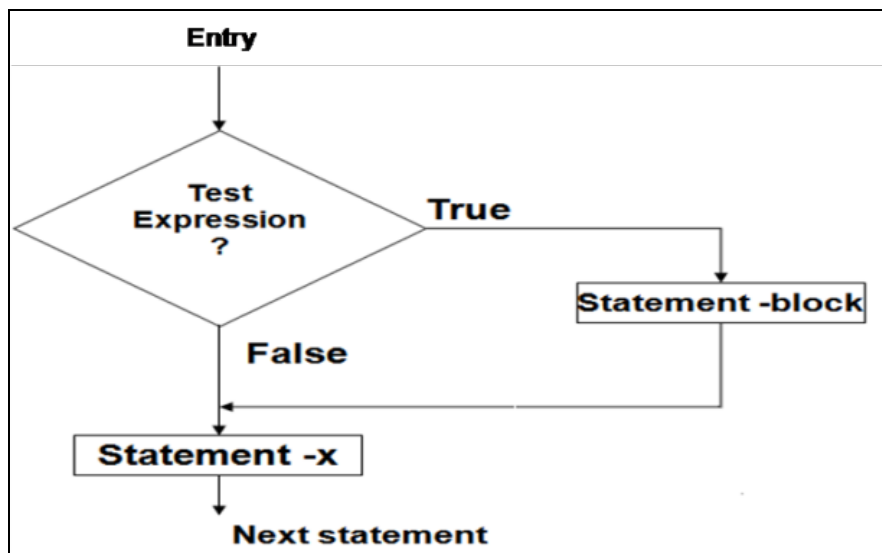
In this lab, student will be able to do C++ programs using

1. simple *if* statement
2. *if-else* statement
3. *switch-case* statement

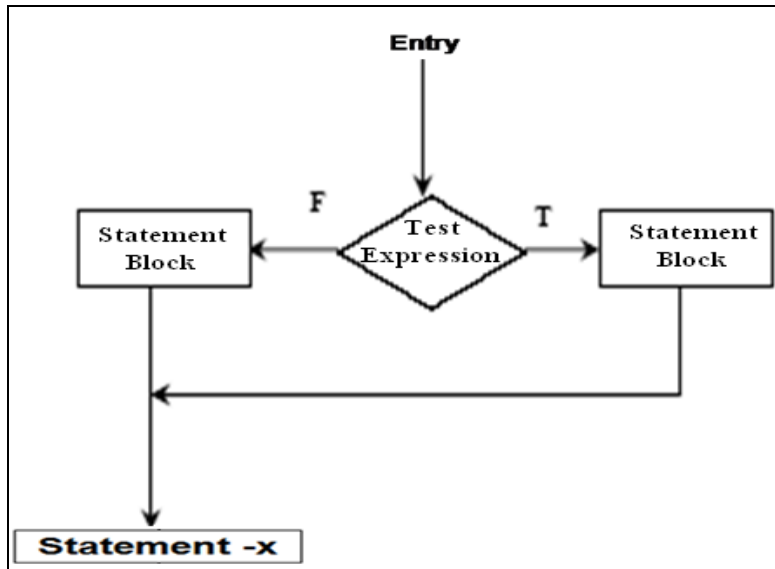
Introduction:

- A control structure refers to the way in which the programmer specifies the order of execution of the instructions

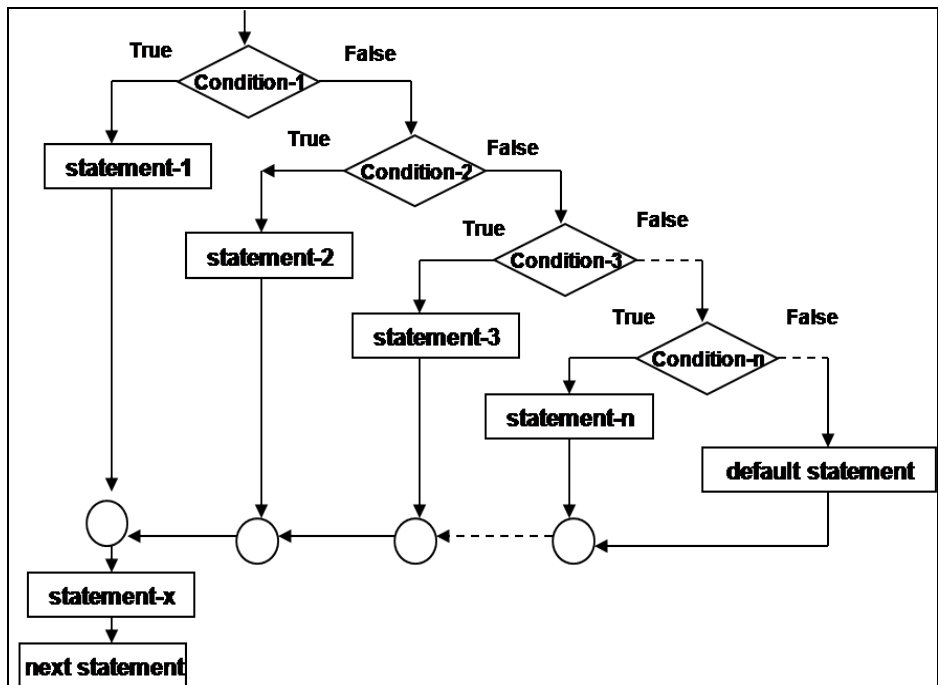
C++ decision making and branching statements flow control actions:

Simple if statement:

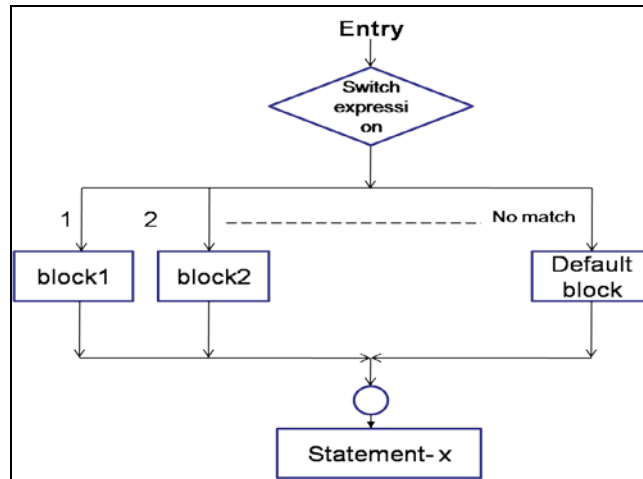
If - else statement:



Else - if ladder:



Switch statement:



Solved exercise

C++ program to compute all the roots of a quadratic equation

```

#include <iostream.h>

#include<math.h>

void main() {

int a,b,c; floatroot1,root2,re,im,disc;

cin>>a>>b>>c;

disc=b*b-4*a*c;

if (disc<0) // first if condition
{

cout<<"imaginary roots\n";

re= - b / (2*a);

im = pow(fabs(disc),0.5)/(2*a);

cout<<re<<" + i"<<im;

cout<<re<<" - i"<<im;

}
}

```

```

else if (disc==0){ //2nd else-if condition

cout<<"real & equal roots";

re=-b / (2*a);

cout<<"Roots are"<<re;

}

else{ /*disc > 0- otherwise part with else*/

cout<<"real & distinct roots";

cout<<"Roots are";

root1=(-b + sqrt(disc))/(2*a);

root2=(-b - sqrt (disc))/(2*a);

cout<<root1<<"and"<<root2;

} }

```

Lab exercises

With the help of various branching control constructs like if, if-else and switch case statements, try doing the following programs.

Write C++ programs to do the following:

1. Check whether the given number is odd or even
2. Find the largest among given 3 numbers
3. Illustrate the LEFT SHIFT and RIGHT SHIFT operations.
4. Compute all the roots of a quadratic equation using *switch case* statement.
[Hint : $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$]

Additional exercises

1. Check whether the given number is zero, positive or negative using else-if ladder.
2. Find the smallest among three numbers using conditional operator.
3. Accept the number of days a member is late to return the book. Calculate and display the fine with the appropriate message using if-else ladder. The fine is charged as per the table below:

Late period	Fine
5 days	Rs. 0.50
6 – 10 days	Rs. 1.00
Above 10 days	Rs. 5.00
After 30 days	Rs. 10.00

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

[OBSERVATION SPACE – LAB 3]

LAB NO:4**CONTROL STRUCTURES - LOOPING****Objectives:**

In this lab, student will be able to:

1. write and execute C++ programs using 'while' statement
2. write and execute C++ programs using 'do-while' statement
3. write and execute C++ programs using 'for' statement

Introduction:

- Iterative (repetitive) control structures are used to repeat certain statements for a specified number of times.
- The statements are executed as long as the condition is true
- These types of control structures are also called as loop control structures
- Three kinds of loop control structures are:
 - while
 - do-while
 - for

C++ looping control structures:**While loop:**

```
while(test condition)
{
    body of the loop
}
```

Do-while loop:

```
do
{
    body of the loop
}
while (test condition);
```

For loop:

```
for (initialization; test condition; increment/decrement)
{
    body of the loop
}
```

Solved exercise

[Understand the working of looping with this illustrative example for finding sum of natural numbers up to 100 with *while* and *do-while* statements]

Using ***do-while***

```
#include <iostream.h>
void main()
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    do
    {
        sum = sum + counter;
        counter = counter +1;
    } while (counter < 100);
    cout<<sum;
}
```

Using ***while***

```
#include <iostream.h>
void main( )
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    while (n<100)
    {
        sum = sum + n;
        n = n +1;
    }
    cout<<sum;
}
```

Lab exercises

With the help of iterative (looping) control structures such as while, do-while and for statements, try doing the following programs.

Write C++ programs to do the following:

1. Reverse a given number and check if it is a palindrome or not.
[Ex: 1234, reverse= $4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 4321$]
2. Generate prime numbers between 2 limits.
3. Check if the sum of the cubes of all digits of an inputted number equals the number itself (Armstrong Number).
4. Generate the multiplication table from numbers up to k terms (using nested for loops).

[Hint: 1 2 3 4 5 K
 2 4 6 8 102*k

 n.....n*K]

Additional exercises

1. Check whether a given number is perfect or not.
 [Hint: Sum of all positive divisors of a given number excluding the given number is equal to the number]Ex: $28 = 1 + 2 + 4 + 7 + 14 = 28$ is a perfect number
2. Evaluate the sine series, $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ to n terms.
3. Check whether the given number is strong or not.
 [Hint: Positive number whose sum of the factorial of its digits is equal to the number itself]
 Ex: $145 = 1! + 4! + 5! = 1 + 24 + 120 = 145$ is a strong number.
4. Find out the generic root of any number.
 [Hint: Generic root is the sum of digits of a number until a single digit is obtained.]
 Ex: Generic root of 456: $4 + 5 + 6 = 15 = 1 + 5 = 6$
5. Generate Floyd's triangle using natural numbers for a given N.
 [Hint: Floyd's triangle is a right angled-triangle using the natural numbers]
 Ex: Input: N = 4
 Output:
 1
 2 3
 4 5 6
 7 8 9 10

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

[OBSERVATION SPACE – LAB 4]

LAB NO: 5**Date:****1D ARRAYS****Objectives:**

In this lab, student will be able to:

- write and execute programs on 1D array

Introduction to 1D Arrays**1 Dimensional Array**

Definition:

- An array is a group of related data items that share a common name.
- The array elements are placed in a contiguous memory location.
- A particular value in an array is indicated by writing an integer number called index number or subscript in square brackets after the array name. The least value that an index can take in array is 0.

Array Declaration:

data-type name [size];

- ✓ where data-type is a valid data type (like int, float, char...)
- ✓ name is a valid identifier
- ✓ size specifies how many elements the array has to contain
- ✓ size field is always enclosed in square brackets [] and takes static values.

Total size of 1D array:

The Total memory that can be allocated to 1D array is computed as:

Total size = size * (sizeof(data_type));

where, size → number of elements in 1-D array

data_type → basic data type.

sizeof() → is an unary operator which returns the size of expression or data type in bytes.

For example, to represent a set of 5 numbers by an array variable n, the declaration the variable n is

int n[5];

Solved exercise

Sample Program code snippet to read n elements into a 1D array and print it:

```
int a[10], i, n;
cout<<"enter no of elements";
cin>>n;
cout<<"\nenter n values\n";
for(i=0;i<n;i++) // input 1D array
    cin>>a[i];
cout<<"\nNumbers entered are:\n";
for(i=0;i<n;i++) // output 1D array
    cout<<a[i]<<endl;
```


Output:

```

enter no of elements
3
enter n values
9
11
13
Numbers entered are:
9
11
13

```

Lab exercises

With the knowledge of 1D and 2D array structures, try doing the following programs.

Write C++ programs to do the following:

1. Arrange the elements in ascending/descending order using Bubble sort method.
2. Insert an element into a 1D array and delete an element from a 1D array.
3. Search an element in a 1D array using linear search.
4. Write a complete C++ program to implement Euler's method for finding an approximate value of y corresponding to x ranging from $[1.0 \text{ to } 1.5]$, with $y_1=5$, $h=0.1$ and $n=6$.
5. Write a complete C++ program to obtain $f(x)$ using Lagrange's Interpolation method for the following records with $x=2.5$

X	0	1	2	3
f(x)	0	2	8	27

Additional exercises on 1D array

1. Find the largest and smallest element in a 1D array.
 2. Print all the prime numbers in a given 1D array.
 3. Search for an element in a given matrix and count the number of its occurrences.
 4. Write a complete C++ program to implement modified Euler's method for finding an approximate value of y corresponding to $x_0=0$, $y_0=1$ and $h=0.05$ with value of last point as 0.1.
 5. Write a complete C++ program to implement Runge-Kutta method for finding an approximate value of y corresponding to $x_0=0$, $y_0=2$ and $h=0.05$ with value of last point as 0.1.
-

[OBSERVATION SPACE – LAB 5]

[OBSERVATION SPACE – LAB 5]

[OBSERVATION SPACE – LAB 5]

[OBSERVATION SPACE – LAB 5]

LAB NO: 6**Date:****2D ARRAYS****Objectives:**

In this lab, student will be able to:

- Write and execute programs on 2D array

Introduction to 2 Dimensional Arrays

- It is an ordered table of homogeneous elements.
- It can be imagined as a two dimensional table made of elements, all of them of a same uniform data type.
- It is generally referred to as matrix, of some rows and some columns. It is also called as a two-subscripted variable.

For example

```
int marks[5][3];
```

```
float matrix[3][3];
```

```
char page[25][80];
```

- The first example tells that marks is a 2-D array of 5 rows and 3 columns.
- The second example tells that matrix is a 2-D array of 3 rows and 3 columns.
- Similarly, the third example tells that page is a 2-D array of 25 rows and 80 columns.

Sample Program code snippet to read and print a 2D array:

```
void main()
{
    int i,j,m,n,a[100][100];
    clrscr();
    cout<<"enter dimension for a:";
    cin>>m>>n;
    cout<<"\n enter elements\n";
    for(i=0;i<m;i++) // input 2D array using 2 for loops
    {
        for(j=0;j<n;j++)
            cin>>a[i][j];
    }
    for(i=0;i<m;i++) // output 2D array with 2 for loops
    {
        for(j=0;j<n;j++)
            cout<<"t"<<a[i][j];
        cout<<"\n";
    }
    getch();
}
```

Lab exercises

With the knowledge of 1D and 2D array structures, try doing the following programs.

Write C++ programs to do the following:

1. Find whether a given matrix is symmetric or not. [Hint: $A = A^T$]
2. Find the trace and norm of a given square matrix.
 [Hint: Trace = sum of principal diagonal elements
 Norm = $\sqrt{\text{sum of squares of the individual elements of an array}}$]
3. Perform matrix multiplication.
4. To interchange the primary and secondary diagonal elements.
5. Interchange any two Rows & Columns in the given Matrix

Additional exercises on 2D array

1. Compute the row sum and column sum of a given matrix.
2. Check whether the given matrix is magic square or not.
3. Check whether the given matrix is a Lower triangular matrix or not.

Ex: 1 0 0
 2 3 0
 4 5 6

[OBSERVATION SPACE – LAB 6]

[OBSERVATION SPACE – LAB 6]

[OBSERVATION SPACE – LAB 6]

[OBSERVATION SPACE – LAB 6]

[OBSERVATION SPACE – LAB 6]

[OBSERVATION SPACE – LAB 6]

LAB NO: 7**Date:****STRINGS****Objectives:**

In this lab, student will be able to:

1. Declare, initialize, read and write the string
2. Write C++ programs with and without string handling functions to manipulate the given string

Introduction

- A string is an array of characters.
- Any group of characters (except double quote sign) defined between double quotations is a constant string.
- Character strings are often used to build meaningful and readable programs.

The common operations performed on strings are

- Reading and writing strings
- Combining strings together
- Copying one string to another
- Comparing strings to another
- Extracting a portion of a string ...

Declaration

Syntax: *char* string_name[size];

- The size determines the number of characters in the string_name.

Solved exercise

Code snippet to read a string

```
void main()
{
    constint MAX = 80; //max characters in string
    charstr[MAX];      //string variable str
    cout<< "Enter a string: ";
    cin>>str;           //put string in str
    cout<< "You entered: " <<str<<endl; //display string from str
}
```

Lab exercises

With the brief introduction and knowledge on string handling functions, try solving these string problems.

Write C++ programs without using STRING-HANDLING functions for the following:

1. Count the number of words in a sentence.
2. Input a string and toggle the case of every character in the input string.

Ex: INPUT: aBcDe

OUTPUT: AbCdE

3. Check whether the given string is a palindrome or not.
4. Arrange 'n' names in alphabetical order (hint: use string handling function-strcmp)

Additional exercises

1. Delete a word from the given sentence.

Ex: INPUT: I AM STUDYING IN MIT

TO BE DELETED: STUDYING

OUTPUT: I AM IN MIT

2. Search for a given substring in the main string.
3. Convert a given string representing a number to an integer.
4. Read a string representing a password character by character and mask every character in the input with '*'.

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

[OBSERVATION SPACE – LAB 7]

LAB NO: 8**Date:****FUNCTIONS****Objectives:**

In this lab, student will be able to:

1. understand modularization and its importance
2. define and invoke a function
3. analyze the flow of control in a program involving function call
4. write programs using functions

Introduction

- A **function** is a set of instructions to carry out a particular task.
- Using functions programs can be structured in a **more modular** way.

Function definition and call

```

// FUNCTION DEFINITION
Return type  Function name  Parameter List
void DisplayMessage(void)
{
    cout << "Hello from function DisplayMessage\n";
}
void main()
{
    cout << "Hello from main";
    DisplayMessage(); // FUNCTION CALL
    cout << "Back in function main again.\n";
}

```

Solved exercise

Code snippet explaining concept of multiple functions

```

void First (void){ // FUNCTION DEFINITION
    cout << "I am now inside function First\n";
}
void Second (void){ // FUNCTION DEFINITION
    cout << "I am now inside function Second\n";
    First(); // FUNCTION CALL
    cout<<"Back to Second\n";
}
void main (){
    cout << "I am starting in function main\n";
    First (); // FUNCTION CALL
    cout << "Back to main function \n";
    Second (); // FUNCTION CALL
    cout << "Back to main function \n";
}

```

Lab exercises

With the knowledge of modularization, function definition, function call etc., write C++ programs which implement simple functions and recursive functions.

Write C++ programs as specified below:

Simple Functions

1. Write a function **Fact** to find the factorial of a given number. Using this function, compute ${}^N C_R$ in the main function.
2. Write a function **IsPrime** to check whether the given number is prime or not. Using this function, generate first N prime numbers in the main function.
3. Write a complete C++ program to obtain the roots of the polynomial $1x^3 + 0x^2 - 1x^1 - 3x^0$, with maximum power of 3, having initial value $x_1=3$ using Newton Raphson's method.
4. Write a complete C++ program to obtain sum of n^{th} term for the series $1+x/1! + x^2/2! + x^3/3! + x^4/4! \dots x^n/n!$, using Taylor's series method.
5. Write a complete C++ program to demonstrate functionality of Newton's Forward difference method for the following records, with $x=2.35$

X	2.0	2.25	2.5	2.75	3
f(x)	9	10.06	11.25	12.56	14

Functions taking (1D/2D) as array Parameter

1. Write a function **Largest** to find the maximum of a given list of numbers. Also write a main program to read N numbers and find the largest among them using this function.
2. Write a function **Sort** to sort a list of names which will use a function **compare** to compare two names. (bubble sort may be used).

Additional exercises

1. Write a function **IsPalin** to check whether the given string is a palindrome or not. Write a main function to test this function.
2. Write a function **CornerSum** which takes as a parameter, no. of rows and no. of columns of a matrix and returns the sum of the elements in the four corners of the matrix. Write a main function to test the function.
3. Write a complete C++ program to find the value of integral using Trapezoidal rule for the following records.

X	7.47	7.48	7.49	7.50	7.51	7.52
f(x)	1.93	1.95	1.98	2.01	2.03	2.06

4. Write a complete C++ program to find the value of integral using Simpson's $1/3^{\text{rd}}$ rule for the following records.

X	0	0.25	0.5	0.75	1
f(x)	1	0.8	0.6667	0.5714	0.5

5. Write a complete C++ program to find the value of integral using Simpson's $3/8^{\text{th}}$ rule for the following records.

X	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f(x)	1.001	1.008	1.027	1.064	1.125	1.216	1.343	1.512	1.729	2.0

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

[OBSERVATION SPACE – LAB 8]

LAB NO: 9**Date:****RECURSIVE FUNCTIONS****Objectives:**

In this lab, student will be able to:

1. Understand concept of recursion
2. Write recursive programs

Introduction

- A recursive function is a function that invokes/calls itself directly or indirectly.

Steps to Design a Recursive Algorithm

- Base case:
 - for a small value of n , it can be solved directly
- Recursive case(s)
 - Smaller versions of the same problem
- Algorithmic steps:
 - Identify the base case and provide a solution to it
 - Reduce the problem to smaller versions of itself
 - Move towards the base case using smaller versions

Solved exercise

Code snippet explaining concept of recursive functions

```
#include <iostream.h>
```

```
long factorial (long a) {
    if (a ==0) //base case
        return (1);
    return (a * factorial (a-1));
}
```

```
void main () {
    long number;
    cout << "Please type a number: ";
    cin >> number;
    cout << number << "! = " << factorial (number);
}
```

Lab exercises

With the knowledge of modularization, function definition, function call etc., write C++ programs which implement recursive functions.

Write C++ programs as specified below:

Recursive Functions

1. Write a recursive function, **GCD** to find the GCD of two numbers. Write a main program which reads 2 numbers and finds the GCD of the numbers using the specified function

Ex: GCD of 9,24 is 3.

2. Write a recursive function **FIB** to generate n^{th} Fibonacci term. Write a main program to print first N Fibonacci terms using function FIB. [Hint: fibonacci series is 0, 1, 1, 2, 3, 5, 8, ...]

Additional exercises

1. Write a recursive function **Print** to print 1 to N numbers using recursion. Do not use loops inside the function. Write a main function to test this.
2. Write a program to multiply two numbers using a recursive function.
[Hint: Multiplication using repeated addition]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

[OBSERVATION SPACE – LAB 9]

LAB NO: 10 INTRODUCTION TO MATLAB

Date:

What is MATLAB?

MATLAB[®] (derived from **Matrix Laboratory**) is a high-level language and interactive environment for numerical computation, visualization, and programming. It originated as an interface to the collections of numerical routines from the LINPACK and EISPACK projects, but it is now a commercial product of **The Mathworks Inc** (www.mathworks.in). The team who developed the software was led by John N. Little and Cleve Moler. MATLAB has evolved into a powerful programming environment containing many built-in functions for doing control systems, signal processing, linear algebra, and other mathematical calculations. The basic MATLAB program is further enhanced by the availability of various toolboxes containing specialized functions. Its strength lies in the fact that complex numerical problems can be solved easily and in a fraction of the time required with other programming language such as C. MATLAB provides an open environment which is extensible and users find it convenient to write new functions whenever the built-in functions fail to do a vital task.

Simulink[®] is a block diagram environment for multi-domain simulation and Model-Based Design. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB[®], enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

MATLAB is available on a number of computing environments: MS Windows, DOS, UNIX workstations, Macintosh etc. For this lab we will be using MATLAB R2013b on Windows platform. It may also be noted that MIT is the first institute in INDIA to implement the Total Academic Headcount (TAH) license of Mathworks products allows for all faculty, staff, and students.

Open Architecture

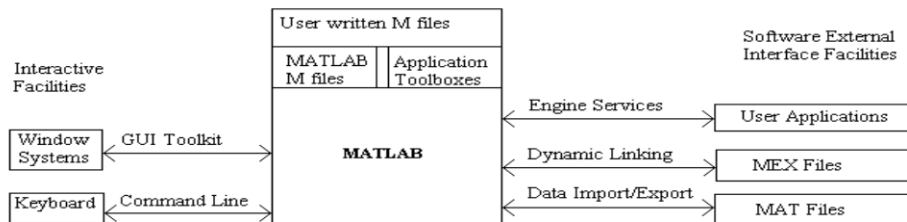


Figure 1

- **Engine Services** : MATLAB is used as computational engine in programs developed in C or Fortran
- **Dynamic Linking** : Runtime linking of C and Fortran routines with MATLAB
- **MATLAB C Compiler**: Generates C code from MATLAB source files.
- **Application Toolboxes**: Collection of MATLAB functions which can be used for various application domains such as Control Systems, Signal Processing, Communication, Image Processing etc.
- **SIMULINK**: Graphical modelling, simulation and prototyping environment integrated with MATLAB.
- **Block sets**: Collection of Simulink blocks for various application domains such as Control Systems, Signal Processing, Communication, Image Processing etc.

Features

- Basic data element: *matrix*
 - auto dimensioning
 - eliminates data type declarations
 - ease of programming
 - operator overloading
 - case sensitive
- Advanced visualisation
 - 2D and 3D
 - simple programming
 - colour graphics
- Open environment

Elementary Matrix Operations

To enter a matrix with real elements

$A = [\begin{matrix} 5 & 3 & 7 \\ 8 & 9 & 2 \\ 1 & 4.2 & 6e-2 \end{matrix}]$ *A is a 3x3 real matrix*

To enter a matrix with complex elements

$B = [\begin{matrix} 5+3j & 7+8j \\ 9+2j & 1+4j \end{matrix}]$ *B is a 2x2 complex matrix*

Transpose of a matrix $A_trans = A'$

Determinant of a matrix $A_det = \det(A)$

Inverse of a matrix $A_inv = \text{inv}(A)$

Matrix multiplication $C = A * A_trans$

Operators

• Arithmetic operators

+	: Addition	-	: Subtraction
*	: Multiplication	/	: Division
\	: Left Division	^	: Power

• Relational operators

<	: Less than	<=	: Less than or equal to
>	: Greater than	>=	: Greater than or equal to
==	: Equal	~=	: Not equal

• Logical operators

&	: AND		: OR	~	: NOT
---	-------	--	------	---	-------

- **Array operations**

Matrix operations preceded by a *.(dot)* indicates array operation.

- **Simple math functions**

sin, cos, tan, asin, acos, atan, sinh, cosh

log, log10, exp, sqrt ...

- **Special constants**

pi, inf, i, j, eps

Control Flow Statements

- **for loop** : for k = 1:m

 end
- **while loop** : while *condition*

 end
- **if statement** : if *condition1*

 else if *condition2*

 else

 end

Some useful commands

who whos clc clf clear load save pause help pwd ls dir

Getting started with MATLAB

- Using **Windows Explorer**, create a folder *user_name* in the directory *c:\cslab\batch_index*
For uniformity, let the *batch_index* be *A1, A2, A3, B1, B2, or B3* and *user_name* be the *roll number*



- Invoke **MATLAB**

Running MATLAB creates one or more windows on your monitor. Of these the **Command Window** is the primary place where you interact with MATLAB. The prompt `>>` is displayed in the Command window and when the Command window is active, a blinking cursor should appear to the right of the prompt.

Interactive Computation, Script files

Objective: Familiarise with MATLAB Command window, do some simple calculations using array and vectors.

Exercises

The basic variable type in MATLAB is a matrix. To declare a variable, simply assign it a value at the MATLAB prompt. Let's try the following examples:

1. Elementary matrix/array operations

To enter a row vector

```
>> a = [5 3 7 8 9 2 1 4]
>> b = [2 6 4 3 8 7 9 5];           % output display suppressed
```

To enter a matrix with real elements

```
>> A = [5 3 7; 8 9 2; 1 4.2 6e-2]
```

To enter a matrix with complex elements

```
>> X = [5+3j 7+8j; 9+2j 1+4j];
```

Transpose of a matrix

```
>> A_trans = A'
```

Determinant of a matrix

```
>> A_det = det(A)
```

Inverse of a matrix

```
>> A_inv = inv(A)
```

Matrix multiplication

```
>> C = A * A_trans
```

Array multiplication

```
>> c = a .* b           % a.*b denotes element-by-element multiplication.
                        % vectors a and b must have the same dimensions.
```

2. Few useful commands

```
>> who           % lists variables in workspace (The MATLAB workspace consists of the
variables you create and store in memory during a MATLAB session.)
```

```
>> whos          % lists variables and their sizes
```

```
>> help inv
```

The online help system is accessible using the help command. Help is available for functions eg. help inv and for Punctuation help punct. A useful command to get started is intro, which covers the basic concepts in MATLAB language. Demonstration programs can be started with the command demo. Demos can also be invoked from the **Help Menu** at the top of the window.

```
>>lookfor inverse
```

The function *lookfor* becomes useful when one is not sure of the MATLAB function name.

```
>>clc %clear command window

>> save session1 % workspace variables stored in session1.mat
>>save myfile.dat a b -ascii % saves a,b in myfile.dat in ascii format

>>clear % clear workspace
>>who

>>load session1 % To load variables to workspace
>>who

>>pwd % present working directory
>>disp(' I have successfully completed MATLAB basics')
```

3. Find the loop currents of the circuit given in Fig.10.1.

Network equations are:

$$\begin{bmatrix} 170 & -100 & -30 \\ -100 & 160 & -30 \\ -30 & -30 & 70 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix}$$

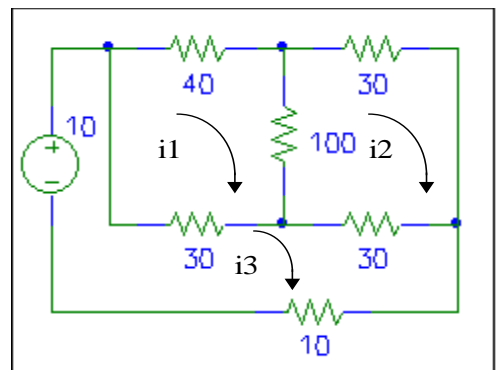


Fig.10.1

Sample Solution a:

(using command line editor)

```
>> Z = [170 -100 -30; -100 160 -30; -30 -30 70];
>> v = [0; 0; 10];
>> i = inv(Z) * v
```

Creating script files

Editing a file: **Home Menu New → Script**, invokes MATLAB Editor/Debugger.

Save file with extension **.m**

Sample Solution b: *(using ex_1b.m file)*

Edit the following commands in the Editor and save as ex_1b.m.

A line beginning with % sign is a comment line.

```
% ex_1b.m
clear; clc;

% Solution of Network equations
Z = [170 -100 -30; -100 160 -30; -30 -30 70];
v = [0; 0; 10];
disp('The mesh currents are : ')
i = inv(Z)*v
```

Typing ex_1b at the command prompt will run the script file, and all the 7 commands will be executed sequentially. The script file can also be executed from the **Run** icon in the MATLAB Editor.

Sample Solution c: *(interactive data input using ex_1c.m)*

```
% ex_1c.m
% Interactive data input and formatted output
clear; clc;

% Solution of Network equations
Z = input('Enter Z : ');
v = input('Enter v : ');
i = Z\v; % Left division - computes inv(Z)*v
disp('The results are : ')
fprintf('i1 = %g A, i2 = %g A, i3 = %g A \n', i(1), i(2), i(3))
```

Results:

```
i =  0.1073
    0.1114
    0.2366
```

4. Find the transient response of the circuit given in Fig.2 (for a period of 5 time constants-5T)

Given:

$$i(t) = \frac{V}{R} e^{-t/T}$$

$$v_c(t) = V \left(1 - e^{-t/T} \right) \quad \text{where } T = RC$$

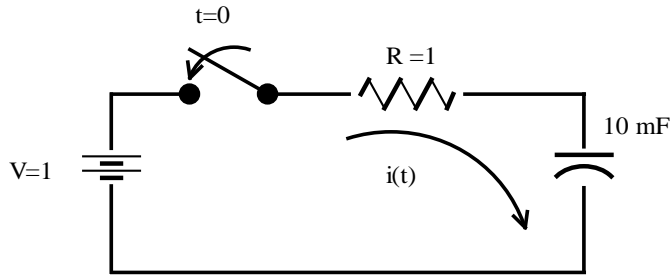


Fig.10.2

Sample Solution a: (illustration of for loop)

```
% ex_5a.m
clear; clc;
disp(' RC transient analysis')
v = input(' Enter source voltage : ');
r = input(' Enter value of resistance : ');
c = input(' Enter value of capacitor: ');
T = r*c;

fprintf('\n The results are : \n\n')
disp('t (sec)      i (A)      v_c (V)')

for n = 1:11
    t(n) = (n-1)*T/2;
    i(n) = (v/r)* exp(-t(n)/T);
    v_c(n) = v*(1 - exp(-t(n)/T));
    fprintf('%6.4f\t%6.4f\t%6.4f\n', t(n),i(n),v_c(n))
end;
```

Colon operator: The *colon operator* is useful for creating index arrays, creating vectors of evenly spaced values, and accessing sub-matrices. A regularly spaced vector of numbers is obtained by means of

$$n = \text{initial value} : \text{increment} : \text{final value}$$

Without the *increment* parameter the default increment is 1. For a 9 x 8 matrix A, A (2,3) is the scalar element located at the 2nd row and 3rd column of A, and a 4x3 sub-matrix can be extracted with A (2:5,1:3). The colon also serves as a wild card i.e., A (2, :) is the 2nd row.

Sample Solution b: (generating vectors)

```
% ex_5b.m
clear; clc;

disp(' RC transient analysis')
v = input(' Enter source voltage : ');
```

```

r = input(' Enter value of resistance : ');
c = input(' Enter value of capacitor : ');
T = r*c;
fprintf('\n The results are : \n\n')
disp('t (sec)      i (A)      v_c (V)')

t = 0 : 0.5*T : 5*T;          % start value : increment : final value
i = (v/r) * exp(-t/T);
v_c = v * (1 - exp(-t/T));

% To output the results in tabular form
A = [t; i; v_c];              % concatenates the vectors
fprintf('%6.4f\t%6.4f\t%6.4f\n', A)

% To write the results to a file

fid = fopen('rctrans.dat','w');
fprintf(fid,'%6.4f\t%6.4f\t%6.4f\n', A)

status = fclose(fid)

```

Results:

t (sec)	i (A)	v_c (V)
0.0000	1.0000	0.0000
0.0050	0.6065	0.3935
0.0100	0.3679	0.6321
0.0150	0.2231	0.7769
0.0200	0.1353	0.8647
0.0250	0.0821	0.9179
0.0300	0.0498	0.9502
0.0350	0.0302	0.9698
0.0400	0.0183	0.9817
0.0450	0.0111	0.9889
0.0500	0.0067	0.9933

5. More Matrix manipulations

A = [1 3 5; 2 4 4; 0 0 3]

B = [1 0 1; 2 2 2; 3 3 3]

Accessing a submatrix A(1:2,2:3); A(1,:); B(:,2)

Concatenating two matrices D = [A B]; E = [A;B];

Adding a row A(4,:) = [1 1 0]

Deleting a column B(:,2) = []

6. Matrix generation functions

zeros(3,3) - 3x3 matrix of zeroes

ones(2,2) - 2x2 matrix of ones

eye(3) - identity matrix of size 3

rand(5) - 5x5 matrix of random numbers

Function File

A *function file* is also an m-file, just like a script file, except it has a function definition line at the top that defines the input and output explicitly.

function<output_list> = *fname* <input_list>

Save the file as *fname.m* The filename will become the name of the new command for MATLAB. Variables inside a function are local. Use *global* declaration to share variables.

7. Create a function file for rectangular to polar conversionSample solution:

```
function [r,theta] = r2p(x)
% r2p - function to convert from rectangular to polar
% Call syntax: [r,theta] = r2p(x)
% Input: x = complex number in the form a+jb
% Output: [r,theta] = the complex number in polar form

r = abs(x);
theta = angle(x)*180/pi;
```


Save the code in a file *r2p.m*.

Executing the function *r2p* in the Command Window

```
>> y = 3+4j;
>> [r,th] = r2p(y)
```

8. Write a function *factorial* to compute the factorial for any integer *n*.

Programming Tips

Writing efficient MATLAB code requires a programming style that generates small functions that are vectorised. Loops should be avoided. The primary way to avoid loops is to use toolbox functions as much as possible.

MATLAB functions operate on arrays just as easily as they operate on scalars. For example, if **x** is an array, then `cos(x)` returns an array of the same size as **x** containing the cosine of each element of **x**.

Avoiding loops - Since MATLAB is an interpreted language, certain common programming habits are intrinsically inefficient. The primary one is the use of *for* loops to perform simple operations over an entire matrix or vector. Wherever possible, you should try to find a vector function that will accomplish the desired result, rather than writing loops.

For example, to sum all the elements of a matrix

By using *for* loops:

```
[Nrows, Ncols]=size(x);
xsum=0.0;
for m=1:Nrows
    for n=1:Ncols
        xsum=xsum+x(m,n);
    end;
end;
```

Vectorised code:

```
xsum=sum(sum(x));
```

Repeating rows or columns - If the matrix has all same values use `ones(M,N)` and `zeros(M,N)`. To replicate a column vector **x** to create a matrix that has identical columns, `x=(12:-2:0)'; X= x*ones(1,11);`. Experience has shown that MATLAB is easy to learn and use. You can learn a lot by exploring the Help & Demos provided and trying things out. One way to get started is by viewing and executing script files provided with the software. Don't try to learn everything at once. Just get going and as you need new capabilities find them, or, if necessary, make them yourself.

OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

[OBSERVATION SPACE – LAB 10]

LAB NO: 11

Date:

MATLAB TUTORIAL 2**Graphics using MATLAB**

MATLAB is capable of producing two dimensional x-y plots and three-dimensional plots, displaying images and even creating and playing movies.

Objective: To learn how to make a simple 2D plot in MATLAB

- **Features**

Advanced visualisation
2D and 3D graphics
ease of programming
color graphics

- **Options**

plot, loglog, semilogx, semilogy, bar, hist, stem, polar, contour

- **Graphic commands**

title, xlabel, ylabel, text, grid, axis, clf, ginput, gtext

- **Line types**

dotted : dashed -- dashdot -.

- **Point types**

point . plus + star * circle o xmark x

- **Color**

red r green g blue b yellow y
magenta m cyan c white w black k

- **Multiple plots**

subplot(mnp) options : 221 211 121

- **Auto scaling**

- **Zooming** zoom on

The Matlab Figure window supports interactive plot editing. Using the mouse objects can be selected and its properties can be then changed. The figures can be saved from the Menu (*filename.fig*).

Exercises

1. To plot $f(t) = \sin(t)$ for $0 \leq t \leq 10$.

Sample Solution:

```
% ex2_1.m
clear; clc; clf;
% To plot f(t) = sin(t)
t = 0 : 0.01 : 10;
y = sin(t);
plot(t,y)
```

Whenever MATLAB makes a plot, it writes the graphics to a **Figure** window. You can have multiple figure windows open, but only one of them is considered the *active* window. The command figure will pop up a new figure window.

2. Plot the function $y = \sin(x)/x$ for $-4\pi \leq x \leq 4\pi$

(The graph is to be labeled, titled, and have grid lines displayed. Try from Tools menu of Figure window)

3. Draw a circle of unit radius

Sample solution:

```
% circle.m
theta = linspace(0,2*pi,100);
x = cos(theta); y = sin(theta);
plot(x,y);
axis('equal');
title('Circle of unit radius');
```

4. Let $f_1(t) = \sin(t)$, $f_2(t) = f_1(t) + 1/3 \sin(3t)$, $f_3(t) = f_2(t) + 1/5 \sin(5t)$, and $f_4(t) = f_3(t) + 1/7 \sin(7t)$. Using multiple plot command subplot, plot the functions $f_1(t)$, $f_2(t)$, $f_3(t)$, $f_4(t)$ for $0 \leq t \leq 2\pi$ in separate axis in the same figure window. Repeat the problem using the command hold to plot the functions $f_1(t)$, $f_2(t)$, $f_3(t)$, $f_4(t)$ in the same figure window.

Sample solution:

% Creating subplots - Illustration

```
t=0:0.01 :2*pi;
f1=sin(t);
f2=f1+(1/3)*sin(3*t);
f3=f2+(1/5)*sin(5*t);
f4=f3+(1/7)*sin(7*t);
v=[0 2*pi -1 1];

subplot(221),plot(t,f1),grid,
```

```

axis(v);
xlabel('f_1 -->');
ylabel('t -->');
subplot(222),plot(t,f2),grid,
axis(v);
xlabel('f_2 -->');
ylabel('t -->');
subplot(223),plot(t,f3),grid,
axis(v);
xlabel('f_3 -->');
ylabel('t -->');
subplot(224),plot(t,f4),grid,
axis(v);
xlabel('f_4 -->');
ylabel('t -->');

```

3D plots

plot3, contour3, surf, mesh, slice, view

5. Use `plot3(x,y,z)` to plot the circular helix $x(t)=\sin(t)$, $y(t)=\cos(t)$ and $z(t)=t$, $0 \leq t \leq 20$.

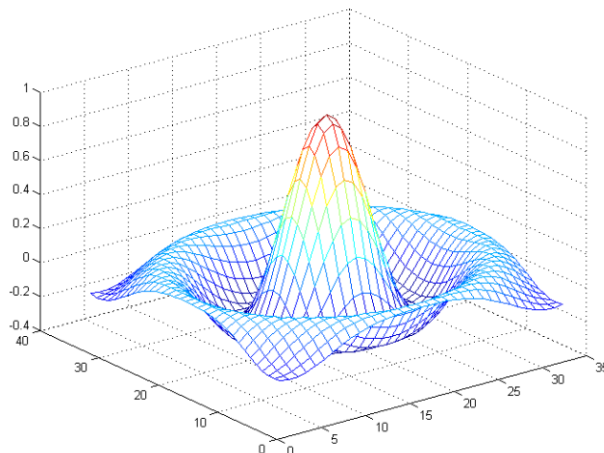
6. 3D Plot of $\sin(r)/r$

Sample Solution :

```

% ex2_6.m
% 3D graphics demo
% 3D Plot of sin(r)/r
[X,Y]=meshgrid( -8 : 0.5 : 8, -8 : 0.5 : 8);
R = sqrt(X.^2 + Y.^2) ;
Z = sin(R)./R;
mesh(Z)

```



Rerun the code using surf(Z) instead of mesh(Z)

GUI using MATLAB(Optional)

Let us try building a simple GUI using MATLAB using the tool *GUIDE*.

- GUIDE stands for Graphical User Interface Development Environment in MATLAB. The GUIDE environment in MATLAB simplifies the manipulation of properties of Graphics Objects and helps us build GUI faster.
 - Learn how to make a simple GUI using GUIDE
 - Try **Help → Creating Graphical user interfaces**
-

[OBSERVATION SPACE – LAB 11]

[OBSERVATION SPACE – LAB 11]

[OBSERVATION SPACE – LAB 11]

[OBSERVATION SPACE – LAB 11]

Modelling with SIMULINK

Objectives: Analysis of simple systems using SIMULINK

- **SIMULINK** - Graphical modelling, dynamic system simulation

1. An RC series circuit with $R = 1\Omega$ & $C = 10\text{mF}$ is connected to a dc source of 10V through a switch. Plot the applied voltage, current and the capacitor voltage for time, $0 \leq t \leq 2\text{s}$, if the switch is closed at $t = 1\text{s}$ & the circuit elements are initially relaxed.

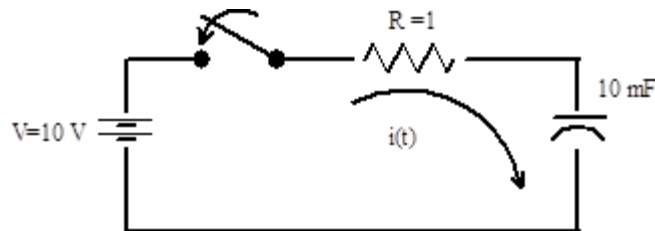


Fig. 12.1

Differential equation defining above circuit

$$V(t) = R i(t) + \frac{1}{C} \int i(t) dt$$

Taking $x = \int i(t) dt$

The above equation may be re-written as

$$R \dot{x} = V(t) - \frac{1}{C} x$$

Block diagram describing the above equation is

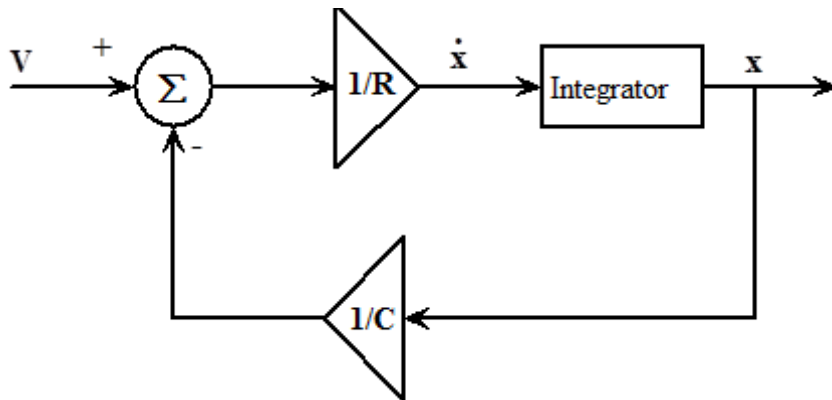


Fig. 12.2

Simulink model describing the above block diagram is

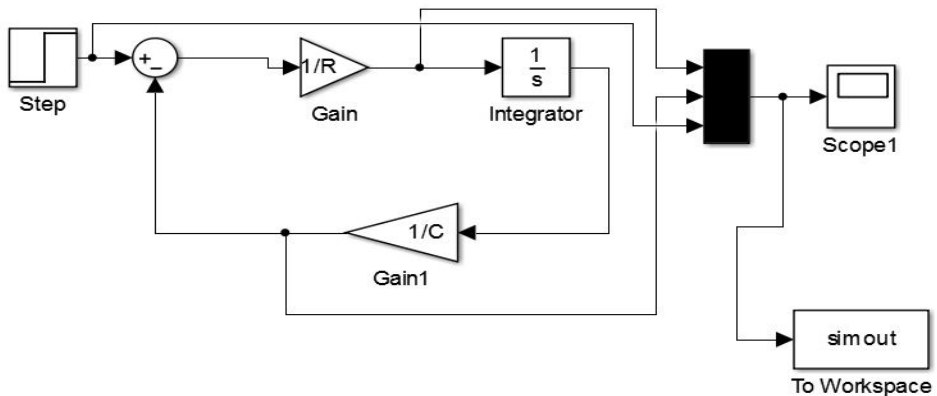


Fig 12.3

- To invoke SIMULINK from MATLAB Command Window
In **Home** menu, Select **New - Simulink Model**
 - Draw the block schematic as shown.
 - Initialize each block by setting appropriate values.
 - Step function applied at $t=1$
 - Set simulation parameters (Start time=0, Stop time =2, Solver Options: Type – Variable step, ode45, Max. step size = 0.01)
 - Run the simulation using the icon provided.
 - Observe the result using Scope
 - Plot results in MATLAB using plot command.

2. An RL series circuit with $R = 2\ \Omega$ & $L = 0.5\ H$ is connected to a dc source of $10V$ through a switch. Plot the applied voltage, current and the capacitor voltage for time, $0 \leq t \leq 5s$, if the switch is closed at $t = 1s$ & the circuit elements are initially relaxed.

$$V(t) = L \frac{di(t)}{dt} + Ri(t) \quad (\text{let } x = i(t), \text{ hence } \dot{x} = \frac{di(t)}{dt})$$

$$L \frac{di(t)}{dt} = V(t) - Ri(t)$$

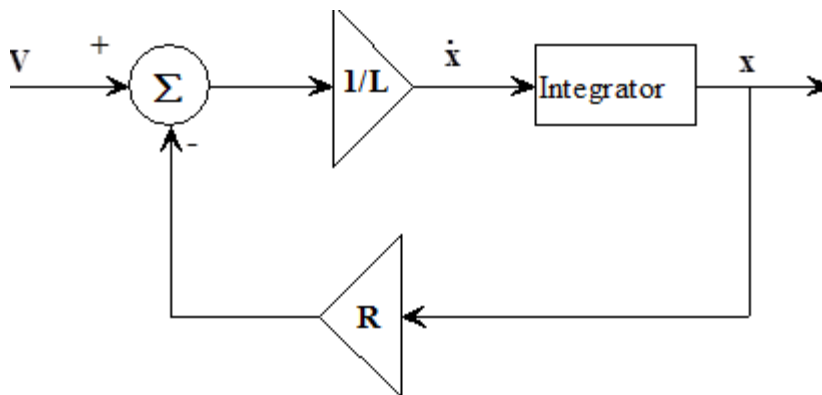


Fig. 12.4

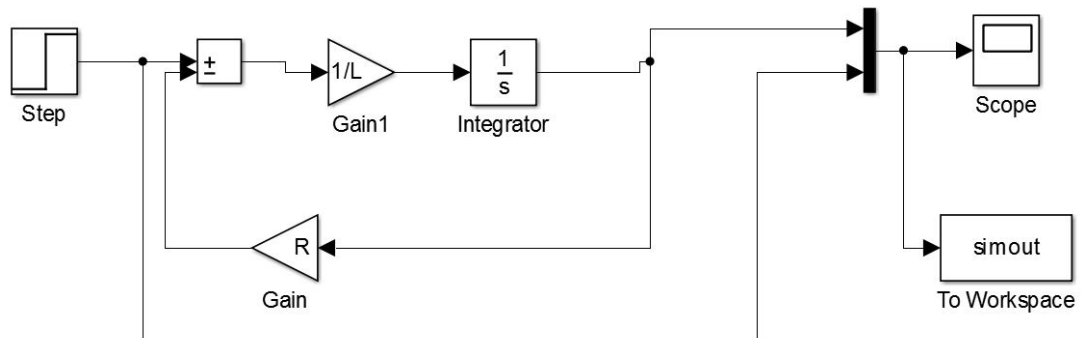


Fig. 12.5

3. An RLC series circuit with $R = 1\Omega$, $L = 1H$, & $C = 10mF$ is connected to a dc source of $10V$ through a switch. Plot the inductor current and the capacitor voltage for time, $0 \leq t \leq 10s$, if the switch is closed at $t = 1s$ & the circuit elements are initially relaxed.

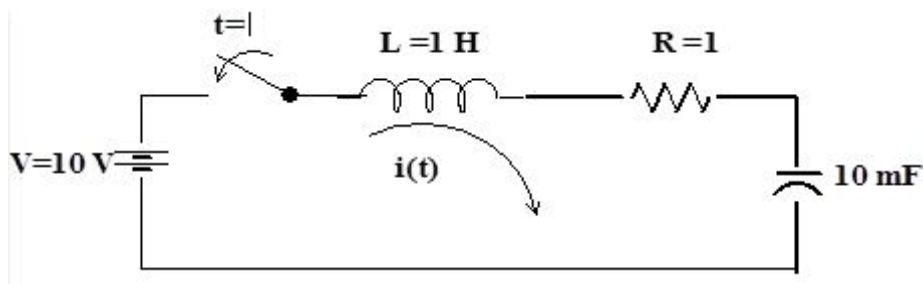


Fig. 12.6

$$V(t) = L \frac{di(t)}{dt} + Ri(t) + \frac{1}{C} \int i(t) dt$$

$$L \ddot{x} = V(t) - R \dot{x} - \frac{1}{C} x$$

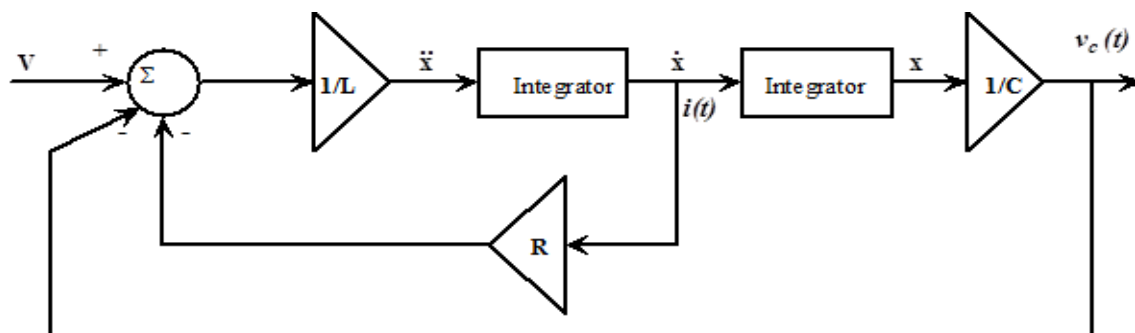


Fig. 12.7

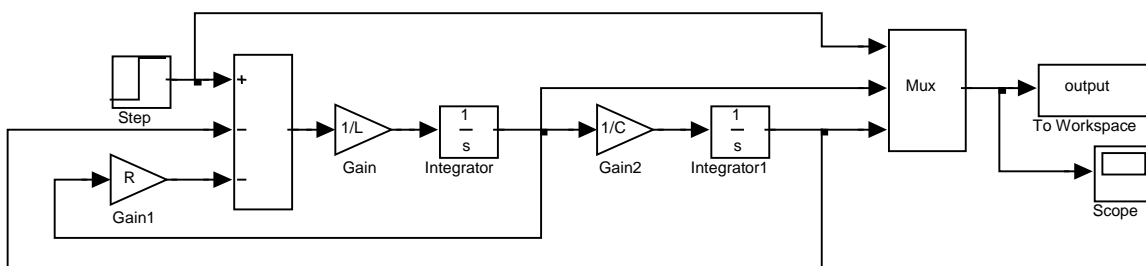


Fig. 12.8

LAB NO: 12

[OBSERVATION SPACE – LAB 12]

LAB NO: 12

[OBSERVATION SPACE – LAB 12]

LAB NO: 12

[OBSERVATION SPACE – LAB 12]

REFERENCES

1. E. Balaguruswamy, “Computing Fundamentals & C Programming”, Tata McGraw Hill, 2008.
2. E. Balaguruswamy, “Object Oriented Programming with C++”, Tata McGraw Hill, 2nd Edition 2007.
3. Yashavant Kanetkar, “Let Us C”, 10th Edition, BPB Publications, 2010.
4. Dr.B.S.Grewal, “Numerical Methods in Engineering & Science”, Khanna publishers, 9th Edition, ISBN: 978-81-7409-248-9, 2010, 8th reprint, 2013.
5. RudraPratap, “Getting started with MATLAB – A Quick Introduction for Scientists and Engineers”, Oxford University Press, ISBN-13:978-0-19-806919-5, 2013.

C++ QUICK REFERENCE

PREPROCESSOR

```
// Comment to end of line
/* Multi-line comment */

#include <stdio.h> // Insert standard header file
#include "myfile.h" // Insert file in current directory
#define X some text // Replace X with some text
#define F(a,b) a+b // Replace F(1,2) with 1+2
#define X \
    some text // Line continuation
#undef X // Remove definition
#if defined(X) // Conditional compilation (#ifdef X)
#else // Optional (#ifndef X or #if !defined(X))
#endif // Required after #if, #ifdef
```

LITERALS

```
255, 0377, 0xff // Integers (decimal, octal, hex)
2147463647L, 0x7fffffff // Long (32-bit) integers
123.0, 1.23e2 // double (real) numbers
'a', '\141', '\x61' // Character (literal, octal, hex)
'\n', '\\', '\'', '\"', // Newline, backslash, single quote, double quote
"string\n" // Array of characters ending with newline and \0
"hello" "world" // Concatenated strings
true, false // bool constants 1 and 0
```

DECLARATIONS

```
int x; // Declare x to be an integer (value undefined)
int x=255; // Declare and initialize x to 255
short s; long l; // Usually 16 or 32 bit integer (int may be either)
char c= 'a'; // Usually 8 bit character
unsigned char u=255; signed char m=-1; // char might be either
unsigned long x=0xffffffffL; // short, int, long are signed
float f; double d; // Single or double precision real (never unsigned)
bool b=true; // true or false, may also use int (1 or 0)
int a, b, c; // Multiple declarations
int a[10]; // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2}; // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={ {1,2,3},{4,5,6} }; // Array of array of ints
char s[]="hello"; // String (6 elements including '\0')
int* p; // p is a pointer to (address of) int
char* s="hello"; // s points to unnamed array containing "hello"
void* p=NULL; // Address of untyped memory (NULL is 0)
int& r=x; // r is a reference to (alias of) int x
enum weekend {SAT, SUN}; // weekend is a type with values SAT and SUN
enum weekend day; // day is a variable of type weekend
enum weekend {SAT=0,SUN=1}; // Explicit representation as int
enum {SAT,SUN} day; // Anonymous enum
typedef String char*; // String s; means char* s;
const int c=3; // Constants must be initialized, cannot assign
const int* p=a; // Contents of p (elements of a) are constant
int* const p=a; // p (but not contents) are constant
const int* const p=a; // Both p and its contents are constant
const int& cr=x; // cr cannot be assigned to change x
```

STORAGE CLASSES

int x;	// Auto (memory exists only while in scope)
static int x;	// Global lifetime even if local scope
extern int x;	// Information only, declared elsewhere

STATEMENTS

x=y;	// Every expression is a statement
int x;	// Declarations are statements
;	// Empty statement
{	// A block is a single statement
int x;	// Scope of x is from declaration to end of
block	
a;	// In C, declarations must precede statements
}	
if (x) a;	// If x is true (not 0), evaluate a
else if (y) b;	// If not x and y (optional, may be repeated)
else c;	// If not x and not y (optional)
while (x) a;	// Repeat 0 or more times while x is true
for (x; y; z) a;	// Equivalent to: x; while(y) {a; z;}
do a; while (x);	// Equivalent to: a; while(x) a;
switch (x) {	// x must be int
case X1: a;	// If x == X1 (must be a const), jump here
case X2: b;	// Else if x == X2, jump here
default: c;	// Else jump here (optional)
}	
break;	// Jump out of while, do, for loop, or switch
continue;	// Jump to bottom of while, do, or for loop
return x;	// Return x from function to caller
try { a; }	
catch (T t) { b; }	// If a throws T, then jump here
catch (...) { c; }	// If a throws something else, jump here

FUNCTIONS

int f(int x, int);	// f is a function taking 2 ints and returning int
void f();	// f is a procedure taking no arguments
void f(int a=0);	// f() is equivalent to f(0)
f();	// Default return type is int
inline f();	// Optimize for speed
f() { statements; }	// Function definition (must be global)

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later. Every program consists of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```
int main() { statements... }    or
int main(int argc, char* argv[]) { statements... }
```

argv is an array of argc strings from the command line. By convention, main returns status 0 if successful, 1 or higher for errors.

EXPRESSIONS

Operators are grouped by precedence, highest first. Unary operators and assignment evaluate right to left. All others are left to right. Precedence does not affect order of evaluation which is undefined. There are no runtime checks for arrays out of bounds, invalid pointers etc.

T::X	// Name X defined in class T
N::X	// Name X defined in namespace N
::X	// Global name X
t.x	// Member x of struct or class t
p → x	// Member x of struct or class pointed to by p
a[i]	// i'th element of array a
f(x, y)	// Call to function f with arguments x and y
T(x, y)	// Object of class T initialized with x and y
x++	// Add 1 to x, evaluates to original x (postfix)
x--	// Subtract 1 from x, evaluates to original x
sizeof x	// Number of bytes used to represent object x
sizeof(T)	// Number of bytes to represent type T
++x	// Add 1 to x, evaluates to new value (prefix)
--x	// Subtract 1 from x, evaluates to new value
~x	// Bitwise complement of x
!x	// true if x is 0, else false (1 or 0 in C)
-x	// Unary minus
+x	// Unary plus (default)
&x	// Address of x
p	// Contents of address p (&x equals x)
x * y	// Multiply
x / y	// Divide (integers round toward 0)
x % y	// Modulo (result has sign of x)
x + y	// Add, or &x[y]
x - y	// Subtract, or number of elements from *x to *y
x << y	// x shifted y bits to left (x * pow(2, y))
x >> y	// x shifted y bits to right (x / pow(2, y))
x < y	// Less than
x <= y	// Less than or equal to
x > y	// Greater than
x >= y	// Greater than or equal to
x == y	// Equals
x != y	// Not equals
x & y	// Bitwise and (3 & 6 is 2)
x ^ y	// Bitwise exclusive or (3 ^ 6 is 5)
x y	// Bitwise or (3 6 is 7)
x && y	// x and then y (evaluates y only if x (not 0))
x r	// x or else y (evaluates y only if x is false(0))
x = y	// Assign y to x, returns new value of x
x += y	// x = x + y, also -= *= /= <=> >>= &= = ^=
x ? y : z	// y if x is true (nonzero), else z
throw x	// Throw exception, aborts if not caught
x, y	// evaluates x and y, returns y (seldom used)

IOSTREAM.H, IOSTREAM

<code>cin >> x >> y;</code>	<code>// Read words x and y (any type) from stdin</code>
<code>cout << "x=" << 3 << endl;</code>	<code>// Write line to stdout</code>
<code>cerr << x << y << flush;</code>	<code>// Write to stderr and flush</code>
<code>c = cin.get();</code>	<code>// c = getchar();</code>
<code>cin.get(c);</code>	<code>// Read char</code>
<code>cin.getline(s, n, '\n');</code>	<code>// Read line into char s[n] to '\n', (default)</code>
<code>if (cin)</code>	<code>// Good state (not EOY)?</code>
	<code>// To read/write any type T:</code>

STRING (Variable sized character array)

<code>string s1, s2= "hello";</code>	<code>//Create strings</code>
<code>s1.size(), s2.size();</code>	<code>// Number of characters: 0, 5</code>
<code>s1 += s2 + ' ' + "world";</code>	<code>// Concatenation</code>
<code>s1 == "hello world";</code>	<code>// Comparison, also <, >, !=, etc.</code>
<code>s1[0];</code>	<code>// 'h'</code>
<code>s1.substr(m, n);</code>	<code>// Substring of size n starting at s1[m]</code>
<code>s1.c_str();</code>	<code>// Convert to const char*</code>
<code>getline(cin, s);</code>	<code>// Read line ending in '\n'</code>
<code>asin(x); acos(x); atan(x);</code>	<code>// Inverses</code>
<code>atan2(y, x);</code>	<code>// atan(y/x)</code>
<code>sinh(x); cosh(x); tanh(x);</code>	<code>// Hyperbolic</code>
<code>exp(x); log(x); log10(x);</code>	<code>// e to the x, log base e, log base 10</code>
<code>pow(x, y); sqrt(x);</code>	<code>// x to the y, square root</code>
<code>ceil(x); floor(x);</code>	<code>// Round up or down (as a double)</code>
<code>fabs(x); fmod(x, y);</code>	<code>// Absolute value, x mod y</code>

