



**MANIPAL INSTITUTE
OF TECHNOLOGY**

MANIPAL

(A constituent unit of MAHE, Manipal)

**DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING**

**CSE 1011
PROBLEM SOLVING USING
COMPUTERS – LAB MANUAL**

**FIRST YEAR
COMMON TO ALL BRANCHES
(2018 CURRICULUM)**

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	Course Objectives and Outcomes	iv	
	Evaluation Plan	iv	
	Instructions to the Students	v-vi	
	Sample Lab Observation Note Preparation	vii-viii	
1	Introduction to Computers and Programming IDE	1	
2	Simple C Programs	6	
3	Branching Control Structures	7	
4	Looping Control Structures	11	
5	1D Arrays	15	
6	2D Arrays	18	
7	Strings	21	
8	Modular Programming and Recursive Functions	23	
9	Pointers, Structures and File Manipulation	27	
10	Matlab Programming 1	32	
11	Matlab Programming 2	42	
12	Matlab Simulink	48	
	References	52	
	C Language Quick Reference	53	

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
- 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
- Demonstrate the programming skills in MATLAB and SIMULINK

Evaluation plan

- Internal Assessment Marks: 60%
 - ✓ Continuous Evaluation (CE) component (for each weeks experiments):10 marks
 - ✓ The CE will depend on punctuality, program execution, maintaining the observation note and answering the questions in viva voce
 - ✓ Total marks of the 12 weeks 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 Class notes, and lab record (King size note book with index) to every lab session
2. Be in time and follow the Instructions from Lab Instructors
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

- Soft copy of the Lab Manual will be provided in the lab machines.
- Follow the instructions on the allotted exercises given in Lab Manual
- 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 record (required till week 4)
- Draw the corresponding flowchart in the Lab record.
 - Flowcharts need not be drawn for the programs from week 5 (1D 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.
 - 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 during 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 at student's end or in a repetition class (if available) 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 later in the manual as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab. Students found using mobile phones and other electronic gadgets will be subjected to rules and regulation of the institution.
- Go out of the lab without permission.

Sample Lab Observation Note Preparation

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 a flow chart and write a 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]

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

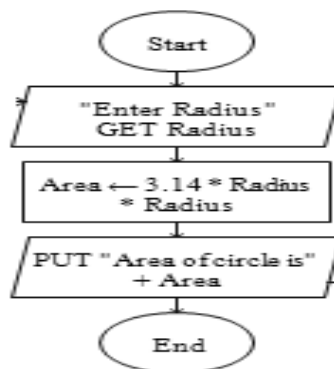
Step 3: [Print the Area]

Print 'Area of a circle =', Area

Step 4: [End of algorithm]

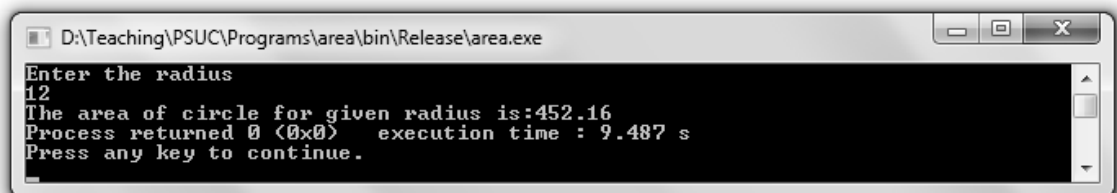
Stop

Flow Chart:



Program:

```
//student name_lab1_1.c
//program to find area of circle
#include<stdio.h>
int main()
{
    int radius;
    float area;
    printf("Enter the radius\n");
    scanf("%d", &radius);
    area=3.14*radius*radius;
    printf("The area of circle for given radius is: %f", area);
    return 0;
}
```

Sample input and output:

The screenshot shows a Windows command prompt window with the title bar "D:\Teaching\PSUC\Programs\area\bin\Release\area.exe". The window contains the following text:

```
Enter the radius
12
The area of circle for given radius is:452.16
Process returned 0 (0x0)   execution time : 9.487 s
Press any key to continue.
```


LAB NO.: 1

INTRODUCTION TO COMPUTERS AND PROGRAMMING IDE

Objectives:

In this lab, student will be able to:

1. Understand the concepts and functions of computer hardware and system software.
2. Understand problem solving & programming paradigms.
3. Explain Algorithms and Flowcharts.
4. Understand different components of a C program.
5. Write, compile and execute simple C programs.

Code: Blocks Integrated Development Environment

Code: Blocks has a C editor and compiler. It allows us to create and test our programs.

Code: Blocks creates Workspace to keep track of the project that is being used. A project is a collection of one or more source files. Source files are the files that contain the source code for the problem.

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.c* program.
 - Open a new notepad file and type the given program
- Save the file with name and extension as "*InchToCm.c*" into the respective directory created.

Sample Program (*InchToCm.c*):

```
// InchToCm.c
#include <stdio.h>
int main()
{
    float centimeters, inches;
```

```

printf( "This program converts inches to centimeters"\n);
printf( "Enter a number");
scanf("%f", &inches);
centimeters = inches * 2.54;
printf("%f inches is equivalent to %f centimeters\n", inches,
centimeters);
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 or through the menu.

PROGRAM STRUCTURE AND PARTS

Comments

The first line of the file is:

```
// InchToCm.c
```

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.c
```

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 lines:

```
#include <stdio.h>
```

This is called a preprocessor directive. It tells the compiler to do something. Preprocessor directives always start with a # sign. The preprocessor directive includes the information in the file *stdio.h*. 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 *int 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 *int* before *main ()* indicates the function is returning an integer 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 *int 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. *printf ()* statement tells the compiler to generate instructions that will display information on the screen when the program run, and *scanf () statement* reads information from the keyboard when the program run.

Format Specifiers	Description	Example
%d	%d is used to print the value of integer variable. We can also use %i to print integer value. %d and %i have same meaning	int v; scanf("%d",&v); printf("value is %d", v);
%f	%f is used to print the value of floating point variable in a decimal form.	float v; scanf("%f",&v); printf("value is %f",v);
%c	%c is used to print the value of character variable.	char ch; scanf("%c",&v); printf("value is %c",ch)
%s	%s is used to print the string	char name[20]; scanf("%s",v); printf("value is %s",name)

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;*

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>
int main()
{
    Variable declarations;
    Executable statements;
} // end main
```

Lab Exercises

1. Write a C program to add two integers a and b read through the keyboard. Display the result using third variable *sum*.
2. Write a C program to convert given number of days into years, weeks and days.
3. Write a C program to find the sum, difference, product and quotient of 2 numbers.
4. Write a C program to print the ASCII value of a character.
5. Write a C program to display the size of the data type int, char, float, double, long int and long double using size of () operator.

LAB NO.: 2

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 to compute simple and compound interest.
[Hint: $SI = PNR/100$, $CI = P(1+R/100)^N - P$]
2. Input radius to 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. Write a C program to evaluate the following expression for the values $a = 30$, $b = 10$, $c = 5$, $d = 15$
(i) $(a + b) * c / d$ (ii) $((a + b) * c) / d$ (iii) $a + (b * c) / d$ (iv) $(a + b) * (c / d)$

Additional Exercises

1. Convert the time in seconds to hours, minutes and seconds. [Hint: 1 hr = 3600 sec]
2. Write a C program for the following
a) $ut + 1/2 at^2$ b) (ii) $a^2 + 2ab + b^2$
3. Determine how much money (in rupees) is in a piggy bank that contains denominations of 20, 10 and 5 rupees 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: $13 * 20 + 11 * 10 + 7 * 5 + 0.50 * 13 = \text{Rs.}411.50$].

LAB NO.: 3

BRANCHING CONTROL STRUCTURES

Objectives:

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

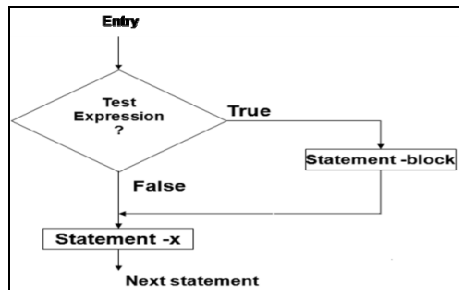
1. **simple *if*** statement
2. ***if-else*** statement
3. ***switch-case*** statement

Introduction:

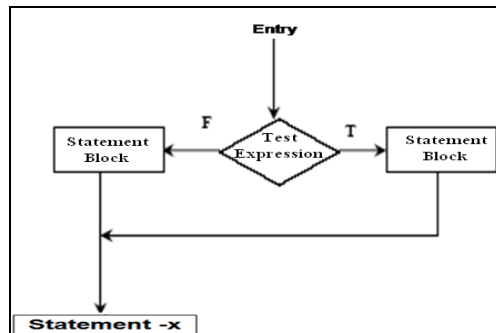
- A control structure refers to the way in which the programmer specifies the order of execution of the instructions

C decision making and branching statements flow control actions:

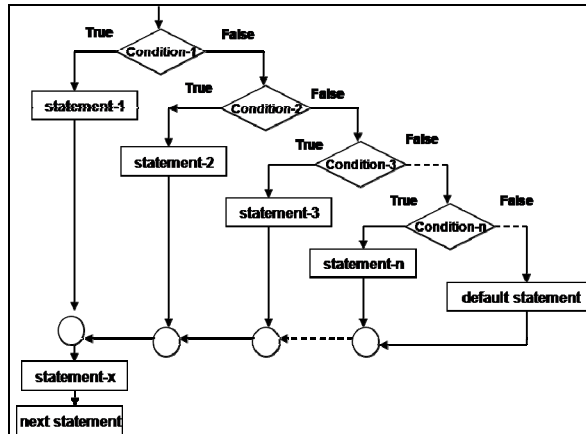
Simple if statement:



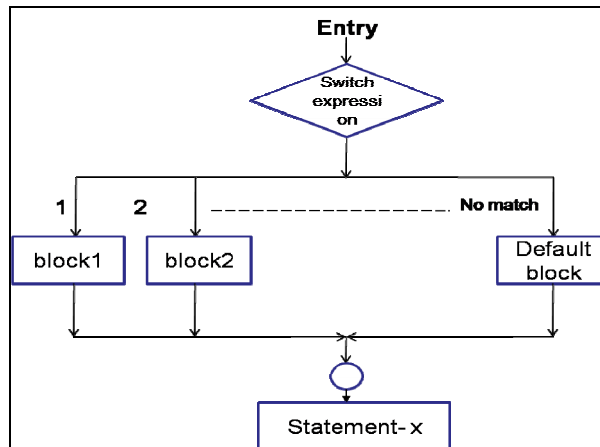
If - else statement:



Else - if ladder:



Switch statement:



Solved Exercise

C program to compute all the roots of a quadratic equation

```
#include<stdio.h>
```

```
#include<math.h>
```

```
int main() {
```

```
int a,b,c;
```

```
float root1, root2, re, im, disc;
```

```
scanf("%d,%d,%d",&a,&b,&c);
```



```

disc=b*b-4*a*c;
if (disc<0) // first if condition
{
printf("imaginary roots\n");
re= - b / (2*a);
im = pow(fabs(disc),0.5)/(2*a);
printf("%f +i %f",re,im);
printf("%f -i %f",re,im);
//printf("%f, re", "+ i",im);
//cout<<re<<"-i"<<im;
}
else if (disc==0){ //2nd else-if condition
printf("real & equal roots");
re=-b / (2*a);
printf("Roots are %f",re);
}
else { /*disc > 0- otherwise part with else*/
printf("real & distinct roots");
printf("Roots are");
root1=(-b + sqrt(disc))/(2*a);
root2=(-b - sqrt(disc))/(2*a);
printf("%f and %f",root1,root2);
}
return 0;
}

```

Lab Exercises

With the help of various branching control constructs like *if*, *if-else* and *switch* case statements,

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. Swap two numbers without using third variable.

4. Compute all the roots of a quadratic equation using **switch case** statement.
[Hint: $x = (-b \pm \sqrt{b^2 - 4ac}) / 2a$]
5. Write a program that will read the value of x and evaluate the following function

$$Y = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases}$$

Use else if statements & Print the result ('Y' value).

6. Find the smallest among three numbers using conditional operator.

Additional Exercises

1. Check whether the given number is zero, positive or negative, using **else-if** ladder.
2. 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

LAB NO.: 4

LOOPING CONTROL STRUCTURES

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 using *while* and *do-while* statements]

Using *do-while*

```
#include<stdio.h>
int main()
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    do
    {
        sum = sum + counter;
        counter = counter +1;
    } while (counter < 100);
    printf("%d",sum);
    return 0;}
```

Using *while*

```
#include<stdio.h>
int main( )
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
```

```

while (n<100)
{
sum = sum + n;
    n = n +1;
}
printf(“%d”,sum);
return 0; }

```

Lab Exercises

With the help of iterative (looping) control structures such as *while*, *do-while* and *for* statements,

Write C programs to do the following:

1. Reverse a given number and check if it is a palindrome or not. (use while loop).
[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 given limits.(use while loop)
3. Check if the sum of the cubes of all digits of an inputted number equals the number itself (Armstrong Number). (use while loop)
4. Generate the multiplication table for ‘*n*’ 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]

5. Generate Floyd’s triangle using natural numbers for a given limit N. (using for loops)

[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

```

6. Evaluate the sine series, $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ to n terms.
7. Write a program using do-while loop to read the numbers until -1 is encountered. Also count the number of prime numbers and composite numbers entered by user.
[Hint: 1 is neither prime nor composite]

Additional Exercises

1. 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.
2. 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 is $4 + 5 + 6 = 15 = 1 + 5 = 6$
3. 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

LAB NO.: 5

1D ARRAYS

Objectives:

In this lab, student will be able to:

- Write and execute programs on 1Dimensional arrays

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 ***Arr***, the declaration the variable ***Arr*** is

int Arr[5];

Solved Exercise

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

```
#include<stdio.h>
int main()
{
    int a[10], i, n;
    printf("enter no of elements");
    scanf("%d",&n);
    printf("enter n values\n");
    for(i=0;i<n;i++) // input 1D array
        scanf("%d",&a[i]);
    printf("\nNumbers entered are:\n");
    for(i=0;i<n;i++) // output 1D array
        printf("%d",a[i]);
    return 0;
}
```

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 arrays,

Write C programs to do the following:

1. Find the largest and smallest element in a 1D array.
2. Print all the prime numbers in a given 1D array.
3. Arrange the given elements in a 1D array in ascending and descending order using bubble sort method. [Hint: use switch case (as case 'a' and case 'd') to specify the order].
4. Insert an element into a 1D array by getting an element and the position from the user.
5. Search the position of the number that is entered by the user and delete that particular number from the array and display the resultant array elements.

Additional Exercises on 1D array

1. Search an element in a 1D array using linear search.
2. Delete all the occurrences of the element present in the array which is inputted by the user.
3. Write a program to enter number of digits and create a number using this digit.
[Hint: Input: Enter number of digits: 3
Enter units' place digit: 1, Enter tens place digit: 2, Enter hundreds place digit: 5
The number is 521]

LAB NO.: 6

2D ARRAYS

Objectives:

In this lab, student will be able to:

- Write and execute programs on 2D dimensional arrays

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.

Solved Exercise:

```
#include<stdio.h>  
int main()  
{  
    int i,j,m,n,a[100][100];  
    printf("enter dimension of matrix");  
    scanf("%d %d",&m,&n);  
    printf("enter the elements");  
    for(i=0;i<m;i++) // input 2D array using 2 for loops  
    {  
        for(j=0;j<n;j++)
```

```

        scanf("%d",&a[i][j]);
    }
    for(i=0;i<m;i++) // output 2D array with 2 for loops
    {
        for(j=0;j<n;j++)
        printf("%d\t",a[i][j]);
        printf("\n");
    }
    return 0;
}

```

Lab Exercises

With the knowledge of 2D arrays,

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 (sum of squares of the individual elements of an array)]
3. Perform matrix multiplication.
4. To interchange the primary and secondary diagonal elements in the given Matrix.
5. Interchange any two Rows & Columns in the given Matrix.
6. Search for an element in a given matrix and count the number of its occurrences.

Additional Exercises on 2D arrays

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

LAB NO.: 7

STRINGS

Objectives:

In this lab, student will be able to:

1. Declare, initialize, read and write a 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 etc.

Declaration

Syntax: *char string_name[size];*

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

Solved Exercise

Program to read and display a string

```
#include<stdio.h>
int main()
{
    const int MAX = 80; //max characters in string
    char str[MAX];      //string variable str
    printf("Enter a string: ");
```

```
scanf("%s",str);
//put string in str
Printf("You entered: %s\n" str); //display string from str
return 0;}
```

Lab Exercises

With the brief introduction and knowledge on strings,

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-*strcpy*)
5. Delete a word from the given sentence.

Ex: INPUT: I AM STUDYING IN MIT
 TO BE DELETED: STUDYING
 OUTPUT: I AM IN MIT

Additional Exercises

1. Search for a given substring in the main string.
2. Delete all repeated words in the given String.
3. Read a string representing a password character by character and mask every character in the input with '*'.
Ex: INPUT: 1234567890
 OUTPUT: *****0
4. Write a C program using 1D array to read an alphanumeric string (Eg. abc14fg67) and count the number of characters and digits in the given string and display the sum of all the digits.

LAB NO.: 8

Modular Programming & Recursive 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
5. Understand concept of recursion
6. Write recursive programs

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.
- 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

Function definition and call

```
// FUNCTION DEFINITION

Return type  Function name  Parameter List
void DisplayMessage(void)
{
    cout << "Hello from function DisplayMessage\n";
}

int main()
{
    cout << "Hello from main";
    DisplayMessage(); // FUNCTION CALL
    cout << "Back in function main again.\n";
    return 0;
}
```

Solved Exercise

1. Program for explaining concept of multiple functions

```
#include<stdio.h>
void First (void){
    printf("I am now inside function First\n");
}
void Second (void){
    printf("I am now inside function Second\n");
    First();
    printf("Back to Second\n");
}
int main (){
    printf("I am starting in function main\n");
    First ();
    printf("Back to main function \n");
    Second ();
    printf("Back to main function \n");
    return 0;
}
```


2. Program to explain the concept of recursive functions

```
#include<stdio.h>
long factorial (long a) {
    if (a ==0) //base case
        return (1);
    return (a * factorial (a-1));
}
int main () {
    long number;
    printf("Please type a number: ");
    scanf("%d",&number);
    printf("%d factorial is %ld",number, factorial (number));
    return 0;
}
```

Lab Exercises

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

Write C programs as specified below:

1. Write a function **Fact** to find the factorial of a given number. Using this function, compute **NCR** in the main function.
2. 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.
3. Write a function **IsPalin** to check whether the given string is a palindrome or not. Write a main function to test this function.
4. 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.
5. 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.

6. 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 function **IsPrime** to check whether the given number is prime or not. Using this function, generate first N prime numbers in the main function.
2. Write a program to multiply two numbers using a recursive function.
[Hint: Multiplication using repeated addition]
3. Write a function **array_sum** to find the sum of 'n' numbers in an array. Write a main program to read 'n' numbers and use **array_sum** function to find the sum of 'n' numbers.
4. Write a function **toggle** to toggle the case of each character in a sentence. Write a main program to read a sentence and use **toggle** function to change the case of each character in the given sentence.

LAB NO.: 9

POINTERS, STRUCTURES AND FILE MANIPULATION

Objectives:

In this lab, student will be able to:

1. Declare and initialize pointer variable
2. Access a variable through its pointer
3. Write basic operations and programs using structures
4. Use basic file handling operations

Introduction:

- A Pointer is a memory location or a variable which stores the address of another variable in memory
- A structure in the C programming language (and many derivatives) is a composite data type (or record) declaration that defines a physically grouped list of variables to be placed under one name in a block of memory.
- A file is a place on disc where group of related data is stored.

Declaring and initializing pointers:

Syntax:

```
data_type * pt_name;
```

This tells the compiler 3 things about the pt_name:

- The asterisk (*) tells the variable pt_name is a pointer variable.
- pt_name needs a memory location.
- pt_name points to a variable of type data_type

Solved Exercise:

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
    int var1 = 11; //two integer variables
```

```

    int var2 = 22;
    int *ptr;           //pointer to integer
    ptr = &var1;        //pointer points to var1
    printf("%d", *ptr);  //print contents of pointer (11)
    ptr = &var2;        //pointer points to var2
    printf("%d", *ptr);  //print contents of pointer (22)
    return 0;
}

```

Declaration and initialization of structures:

Déclaration :

```

struct student
{
    int rollno, age;
    char name[20];
} s1, s2, s3;

```

Initialisation :

```

int main( ){
    struct
    {
        int rollno;
        int age;
    } stud={20, 21};
    ...
    ...
    return 0;
}

```

Solved Exercise:

```

#include<stdio.h>
struct Book{           //Structure Definition
    char title[20];
    char author[15];
    int pages;
    float price;
};

```

```

int main( ){
struct Book b[10];
int i,j;
printf("Input values");
for (i=0;i<10;i++){
scanf("%s %s %d %f",b[i].title,b[i].author,&b[i].pages,&b[i].price);
}
for (j=0;j<10;j++){
printf("title %s\n author %s\n pages %d\n
price%f\n",b[j].title,b[j].author,b[j].pages,b[j].price);
}
return 0;
}

```

Declaration and initialization of structures:

Declaration:

FILE *fp;

Here FILE refers to the type of name for file description and it is user defined data type defined in **stdio.h**

fp refers to the file pointer.

File functions:

1. fopen(): This function open a file.
 fp = fopen("filename", "mode");
 Mode can be
 - **r** open file for reading only
 - **w** open file for writing only
 - **a** open file for appending (adding) data
2. fclose() : This function is used to close an active file.
 close(fp);

Solved Exercise:

```
#include <stdio.h>

void main()
{
    FILE *fp1;
    char c;
    fp1= fopen("INPUT", "w"); /* open file for writing */
    while((c=getchar()) != EOF) /*get char from keyboard until CTRL-Z*/
        putchar(c,fp1);          /*write a character to INPUT */
    fclose(fp1);                 /* close INPUT */
    fp1=fopen("INPUT", "r");      /* reopen file */
    while((c=getc(fp1))!=EOF) /*read character from file INPUT*/
        printf("%c",c);          /* print character to screen */
    fclose(fp1);
}
```

Lab Exercises

With the knowledge of pointer, Structures and files

Write C programs as specified below:

- 1) Find the length of the string using pointers.
- 2) Find the maximum number in the input integer array using pointers.
- 3) Create a student record with name, rollno, marks of 3 subjects (m1, m2, m3).
Compute the average of marks for 3 students and display the names of the students in ascending order of their average marks.
- 4) Create an employee record with emp-no, name, age, date-of-joining (year), and salary. If there is 20% hike on salary per annum, compute the retirement year of each employee and the salary at that time. [standard age of retirement is 55]
- 5) Create a file and store details of "n" students and display the list of students.
- 6) Copy the contents of one file into another.

Additional Exercises

1. Write a C program to find the frequency of a word in an inputted string using pointers
2. Write a C program to create a structure to store the following information of an employee.
Emp_no, name, pay, date-of-join.
It is decided to increase the pay as per the following rules:
 - Pay ≤ Rs. 2000; 15% increase
 - Pay ≤ Rs. 5000 but > Rs 2000: 10% increase
 - Pay > Rs. 5000 no increase.
3. Write a C program to create a text file and convert the file content to uppercase.

LAB NO.: 10

MATLAB PROGRAMMING 1

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems.

Some of the areas in which **toolboxes** are available include

- **Signal processing:** This Toolbox provides functions and applications to generate, measure, transform, filter, and visualize signals.
- **Control systems:** Provides industry-standard algorithms and applications for systematically analyzing, designing, and tuning linear control systems.
- **Neural networks:** Provides functions and apps for modeling complex nonlinear systems that are not easily modelled with a closed-form equation. Neural Network Toolbox supports supervised learning with feed-forward, radial basis, and dynamic networks. It also supports unsupervised learning with self-organizing maps and competitive layers
- **Fuzzy logic:** This toolbox lets you model complex system behaviors using simple logic rules, and then implement these rules in a fuzzy inference system.
- **Wavelets:** Provides functions and an application for developing wavelet-based algorithms for the analysis, synthesis, denoising, and compression of signals and images. The toolbox lets you explore wavelet properties and applications such as speech and audio processing, image and video processing, biomedical imaging, and 1-D and 2-D applications in communications and geophysics.

The MATLAB System

The MATLAB system consists of five main parts:

- **Development Environment:** This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, an editor and debugger, and browsers for viewing help, the workspace, files, and the search path.
- **The MATLAB Mathematical Function Library:** This is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.
- **The MATLAB Language:** This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick and dirty throw-away programs, and “programming in the large” to create large and complex application programs.

- **Graphics:** MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.
- **The MATLAB External Interfaces/API:** This is a library that allows you to write C and Fortran programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

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

$A = [5 \ 3 \ 7; 8 \ 9 \ 2; 1 \ 4.2 \ 6e-2]$ *A is a 3x3 real matrix*

To enter a matrix with complex elements

$B = [5+3j \ 7+8j; 9+2j \ 1+4j]$ *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 e.g. 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. *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) = []
```

4. *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
```

5. *Find the loop currents of the circuit given in Fig.10.1.*

Network equations are:

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

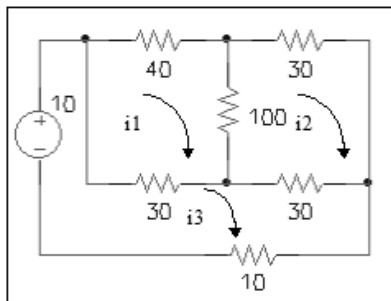


Fig. 10.1

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
```

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 conversion

Sample 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.

MATLAB PROGRAMMING 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 labelled, titled, and have grid lines displayed. Try from Tools menu of Figure window)

3. 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

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

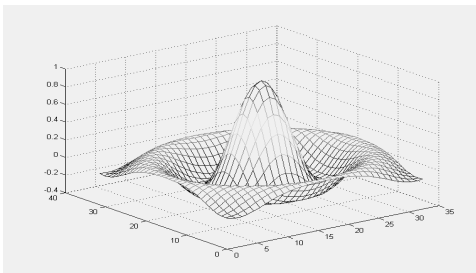
5. 3D Plot of $\sin(r)/r$

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 $\text{surf}(Z)$ instead of $\text{mesh}(Z)$

MATLAB for Signal Processing

MATLAB offers Signal Processing toolbox that provides functions and apps to generate, measure, transform, filter, and visualize signals. The toolbox includes algorithms for resampling, smoothing, and synchronizing signals, designing and analyzing filters, estimating power spectra, and measuring peaks, bandwidth, and distortion. The toolbox also includes parametric and linear predictive modeling algorithms. You can use Signal Processing Toolbox to analyze and compare signals in time, frequency, and time-frequency domains, identify patterns and trends, extract features, and develop and validate custom algorithms to gain insight into your data.

Signal Processing and Visualization

Signal processing toolbox can be used to generate and visualize widely used periodic and aperiodic waveforms and other kinds of waveforms. The following example shows how to generate Periodic waveform.

Periodic waveforms:

The toolbox can be used to produce periodic signals such as sawtooth and square.

- The sawtooth function generates a sawtooth wave with peaks at ± 1 and a period of 2π . An optional width parameter specifies a fractional multiple of 2π at which the signal's maximum occurs.
- The square function generates a square wave with a period of 2π . An optional parameter specifies duty cycle, the percent of the period for which the signal is positive.

Aperiodic Waveforms:

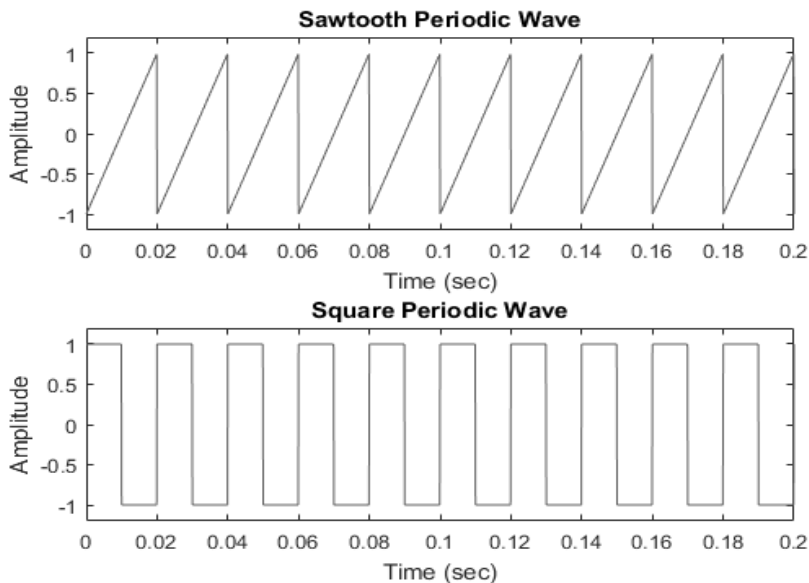
To generate triangular, rectangular and Gaussian pulses, the toolbox offers the tripuls, rectpuls and gausspuls functions.

- The tripuls function generates a sampled aperiodic, unity-height triangular pulse centered about $t = 0$ and with a default width of 1.
- The rectpuls function generates a sampled aperiodic, unity-height rectangular pulse centered about $t = 0$ and with a default width of 1.

6. *To generate 1.5 seconds of a 50 Hz sawtooth (respectively square) wave with a sample rate of 10 kHz.*

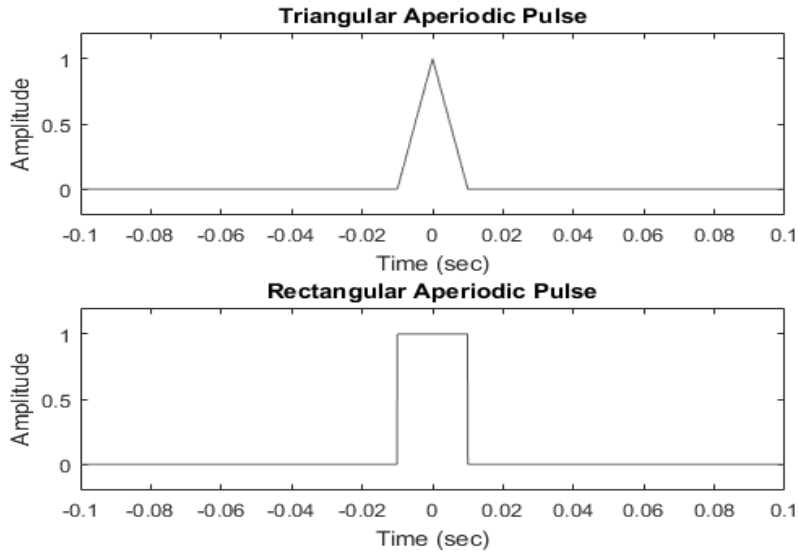
Sample Solution:

```
fs = 10000;  
t = 0:1/fs:1.5;  
x1 = sawtooth(2*pi*50*t);  
x2 = square(2*pi*50*t);  
subplot (211), plot (t, x1), axis([0 0.2 -1.2 1.2])  
xlabel ('Time (sec)'); ylabel ('Amplitude');  
title ('Sawtooth Periodic Wave');  
subplot (212)  
plot(t, x2)  
axis([0 0.2 -1.2 1.2]);  
xlabel('Time (sec)'); ylabel('Amplitude');  
title('Square Periodic Wave')
```



7. To generate 2 seconds of a triangular (respectively rectangular) pulse with a sample rate of 10 kHz and a width of 20 ms. (Hint: Use tripuls and rectpuls functions)

Sample Output:



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.

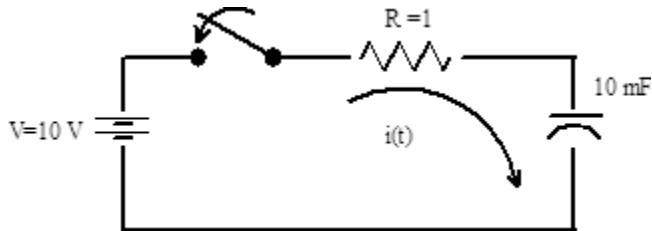


Fig. 12.1

Differential equation defining above circuit

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

$$\text{Taking } x = \int i(t) dt$$

The above equation may be re-written as

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

Block diagram describing the above equation is

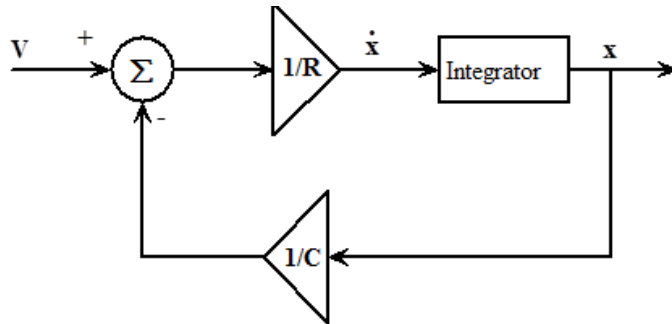


Fig. 12.2

Simulink model describing the above block diagram is

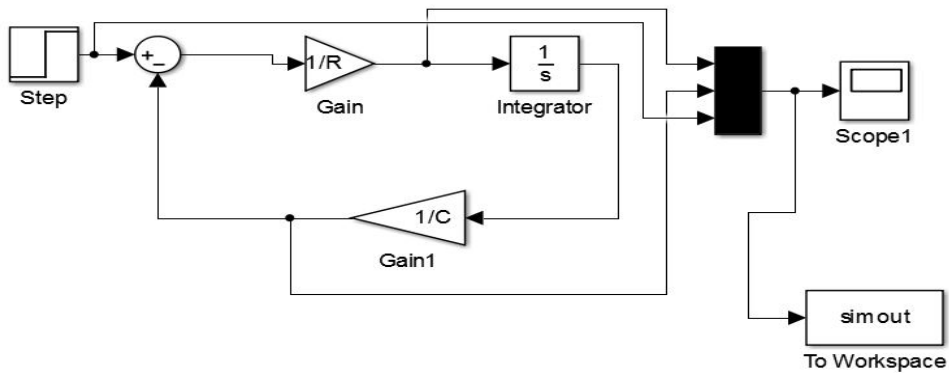


Fig. 12.3

- 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.

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

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

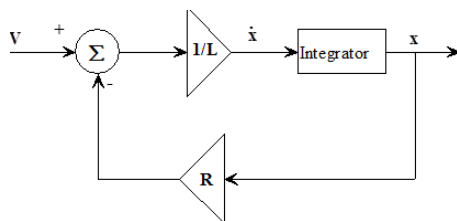


Fig. 12.4

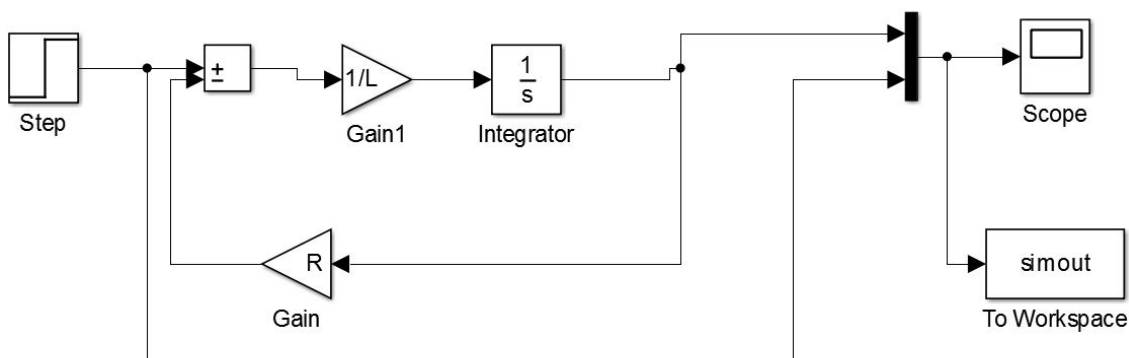


Fig. 12.5

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.

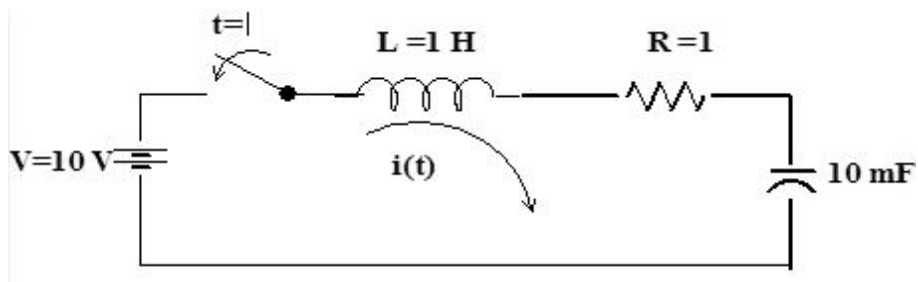


Fig. 12.6

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

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

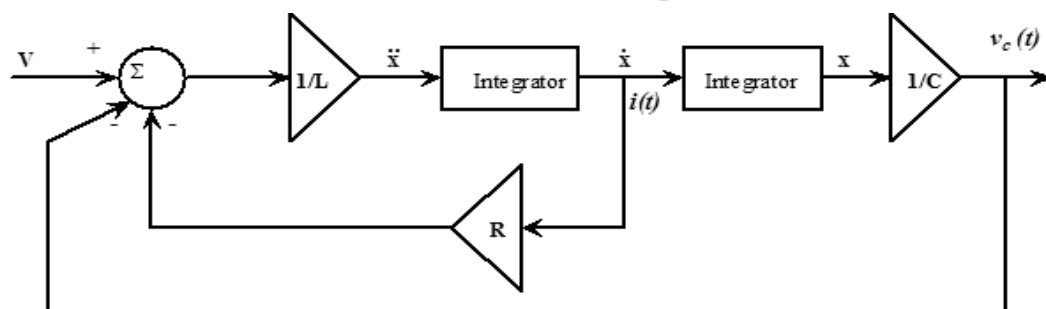


Fig. 12.7

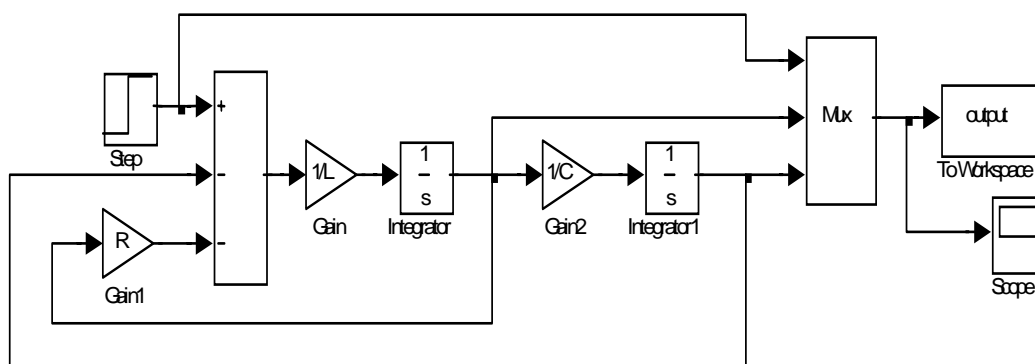


Fig. 12.8

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. Rudra Pratap, “Getting started with MATLAB – A Quick Introduction for Scientists and Engineers”, Oxford University Press, ISBN-13:978-0-19-806919-5, 2013.

C LANGUAGE 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
#ifdef 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)

STDIO.H.H, STDIO.H

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

STRING (Variable sized character array)

string s1, s2= "hello";	//Create strings
s1.size(), s2.size();	// Number of characters: 0, 5
s1 += s2 + ' ' + "world";	// Concatenation
s1 == "hello world";	// Comparison, also <, >, !=, etc.
s1[0];	// 'h'
s1.substr(m, n);	// Substring of size n starting at s1[m]

<code>sl.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>
