MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CSE 1181: PROGRAMMING FOR PROBLEM SOLVING - LAB MANUAL

FIRST YEAR B.TECH. CURRICULUM [CSE STREAM]

(2024 CURRICUL UM)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CONTENTS

Course Objectives: This laboratory course enable students to

- Understand the basics of computing and various problem solving techniques
- Understand and use various programming concepts using C language
- Understand the concepts of modular programming and implement some mathematical applications
- Understand and implement Structures, Pointers and Files

Course Outcomes

On completion of this laboratory course, the students will be able to:

1. Develop algorithms and implement programs to solve basic computational problems using C language.
2. Utilize arrays and functions to organize, manipulate data efficiently, and to achieve code reusability.
3. Implement structures and pointers for solving complex problems and file handling techniques to ensure data persistence and retrieval.

Evaluation plan

- Internal Assessment Marks: 60%
  - ✓ Students must work out the C programs in code blocks only
  - ✓ Continuous Evaluation (CE) component (for 3 times during semester): 12 marks each
  - ✓ The CE will depend on punctuality, program execution, maintaining the observation and record note and answering the questions in viva voce
  - ✓ Mid Sem Lab Exam for 2 -hours : 24 marks

- 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 observation notes (Long note book with index) to every lab session.
2. Be on 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

- 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 an algorithm for the first lab program of all the lab sheets in the Lab record
- Draw the corresponding flowchart in the Lab record.
  - Flowcharts need be drawn for the first lab program of all the lab sheet.

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

1. 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.
2. Go out of the lab without permission.

<u>Sample Lab Observation Note Preparation</u>

Title: SIMPLE C PROGRAMS

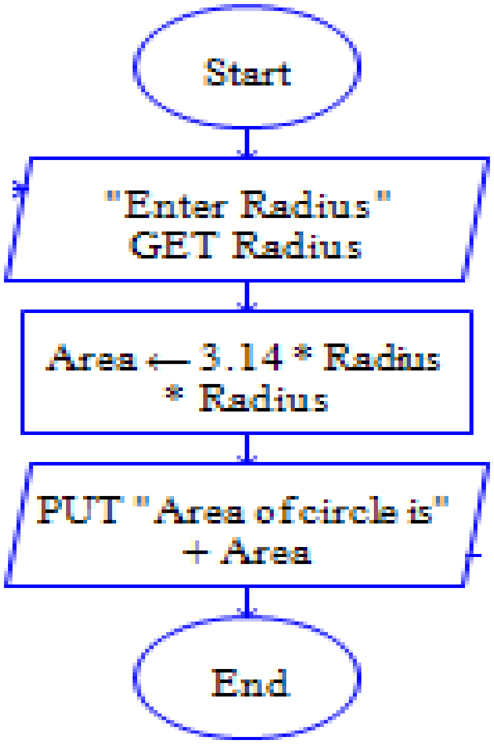
1. Program to find area of the circle. (Hint: Area=3.14*r*r)

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]
                Area  3.1416 * radius * radius
Step 3: [Print the Area]
                Print 'Area of a circle =', Area
Step 4: [End of algorithm]
                Stop
```

Flow Chart:

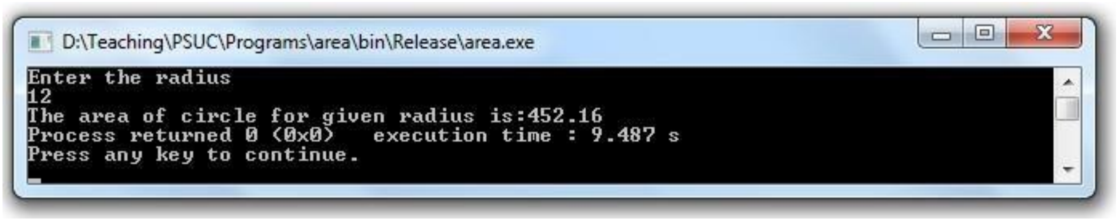


Program:

```
//student name_lab0_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: Screen shot of result – must be included.



```
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.
```

PROGRAM STRUCTURE AND PARTS

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, radius*;* is called a variable declaration. This line tells the compiler to reserve places in memory with adequate size for a int number (the *int* keyword indicates the variable as an integer number). The programs often have many different variables of many different types.

EXECUTABLESTATEMENTS

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 compilr 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)

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
```

SIMPLE PROGRAM

LAB NO 1:

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.

CODE BLOCKS

1. Students shall do all the programming exercises in CODE BLOCKS only. The URLfor downloading and procedure is given below.

Procedure for CODE BLOCKS downloading and installing.

Code:Blocks is a free C/C++ and Fortran IDE built to meet the most demanding needsof its users. It is designed to be very extensible and fully configurable.

Code::Blocks Integrated Development Environment

Code::Blocks is a free, open-source, cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It's highly configurable and extensible, making it a popular choice for C and C++ development on the Ubuntu Linux platform.

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.

Code::Blocks will help in editing, storing, compiling and executing C programs.

Creating and Editing a C Program

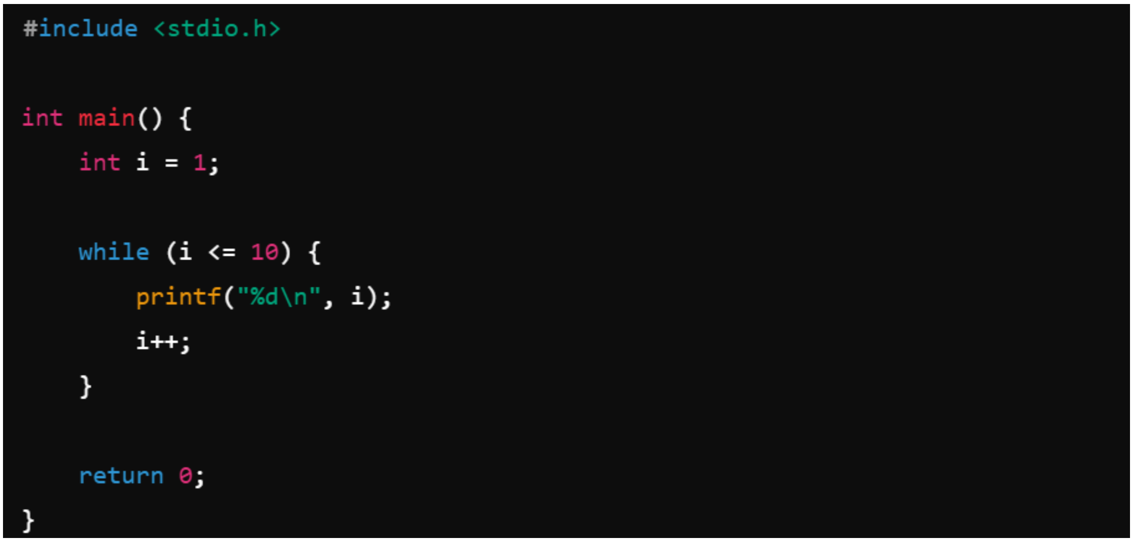
1. **Open Code::Blocks**: You can open it from the application menu or by typing codeblocks in the terminal.

2. **Create a New Project**:

- Go to File → New→ Project.
- Select Console application and click Go.
- Choose ‘C’ as the language and click Next.
- Follow the wizard to set the project title and location.

3. **Add a C Source File**:
    - Right-click on the Sources folder in the Projects tab.
    - Select Add file and choose New file.
    - Select C/C++ source and follow the prompts to create a new .c file.

Sample Program



```
#include <stdio.h>

int main() {
    int i = 1;

    while (i <= 10) {
        printf("%d\n", i);
        i++;
    }

    return 0;
}
```

1. **Write the Code**: Open the .c file you created and write the above code.
2. **Save the File**: Go to File → Save or press Ctrl+S.

Compiling and Executing the Program

1. **Build the Project**:
    - Go to Build → Build or press F9.
    - The build log will show the compilation process.
    - Ensure there are no compilation errors.

2. **Run the Program**:

- Go to Build → Run or press F10.
- The output will appear in the terminal window within Code::Blocks.

URL: **https://www.codeblocks.org/downloads/binaries/**
https://www.codeblocks.org/downloads/

In this URL you may be able to download for the following operating systems.

- **Windows XP / Vista / 7 / 8.x / 10**
- **Linux 32 and 64-bit**
- **Mac OS X**

Please look into the below YouTube link which details how to download and installcode blocks for windows 10.

YouTube: https://www.youtube.com/watch?v=GWJqsmitR2I

Lab Exercise

1. Write a C program to add two integers a and b read through the keyboard. Displaythe result using third variable sum
2. Write a C program to find the sum, difference, product and quotient of 2 numbers.
3. Write a C program to print the ASCII value of a character
4. Write a C program to display the size of the data type int, char, float, double, long int and long double using sizeof( ) operator.
5. Write a C program to input P, N and R, compute and display simple and compound

interest. [Hint: SI = PNR/100, CI = $P(1+R/100)^N-P$]

6. Write a C program to input radius, compute and display the volume and surface area of a sphere. [Hint: volume = $(4\pi r^3)/3$, Area=$4\pi r^2$]

7. Write a C program to convert the given temperature in Fahrenheit to Celsius [Hint: C=5/9(F-32)]

8. 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 Exercise

1. Write a C program to convert given number of days into years, weeks and days.
2. Write a C program to convert the time in seconds to hours, minutes and seconds. [Hint: 1 hr =3600 sec]
3. Write a C program for the following

   (i) $ut + 1/2\ at^2$ (ii) $a^2 + 2ab + b^2$
4. 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 = Rs.411.50].

Debugging exercise

Instructions: Identify and correct the syntax or logical errors in the code.

Task1: The following C program is for swapping two numbers without using a temporary variable.

```
#include <stdio.h>
int main() {
   int x, y;
   printf("Enter two integers: );
   scanf("%d %d", &x, &y);
   x = x + y;
   y = x - y;
   x = x - y
   printf("After swapping: x = %d, y = %d\n", x, y);
   return 0;
}
```

Task 2: The following C program is to calculate the square of a given number.

```
#include <stdio.h>
int main() {
   int num, square;
   printf("Enter an integer: ";
   scanf("%d", num);
```

```
    square = num * num;
    printf("Square of %d = %d\n", num, square);
    return 0;
}
```

Task 3: The following C program is to convert a given distance in kilometers to miles. (1 kilometer = 0.621371 miles)

```
#include <stdio.h>
int main() {
    double kilometers, miles;
    printf("Enter distance in kilometers: ");
    scanf("%lf", &kilometers);
    miles = kilometers * 0.621371;
    printf("%lf kilometers is equal to %lf miles\n", &kilometers, miles);
    return 0;
}
```

LAB NO.: 2

BRANCHING CONTROL STRUCTURES

Objectives:

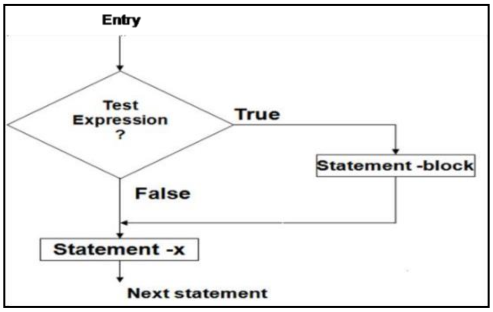
In this lab, student will be able to do C programs using

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

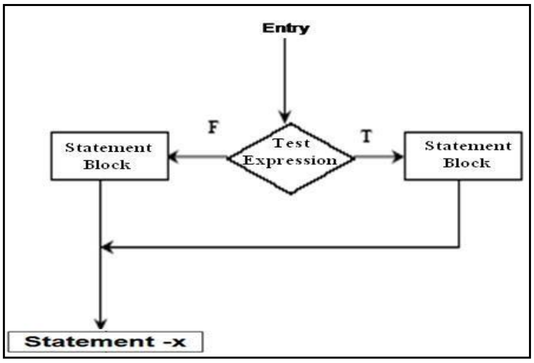
Introduction:

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

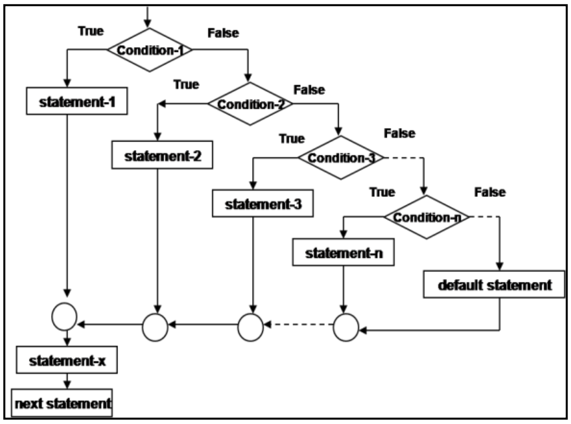
Simple if statement:



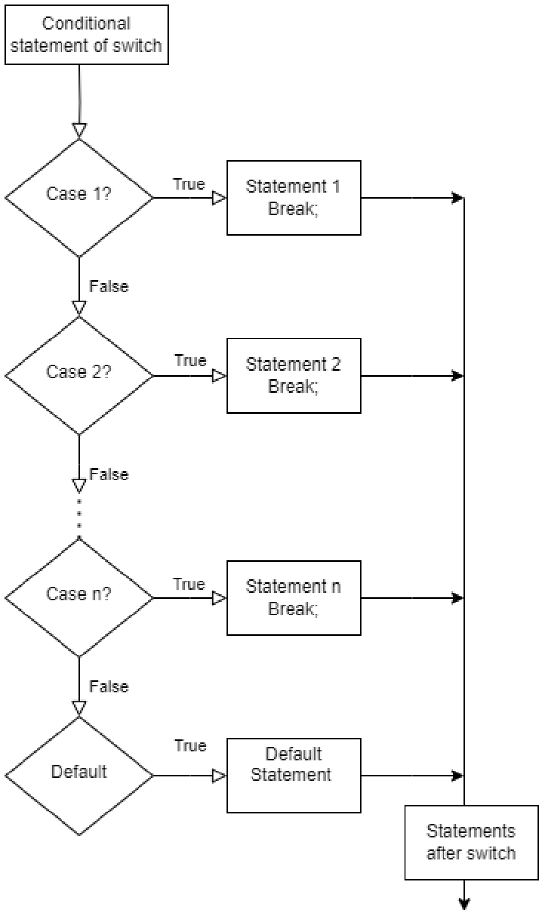
If - else statement:



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);
}
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 Exercise

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. Compute all the roots of a quadratic equation using ***switch case*** statement.
   [Hint: x = (-b +/- sqrt(b^2-4ac))/2a]
4. Find the smallest among three numbers using conditional operator.

Additional Exercise

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

3. 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).

Debugging exercise

Instructions : Identify and correct the syntax or logical errors in the code.

Task 1: A C program to print the day based on the number (1 for Monday, 2 for Tuesday, etc.) using switch statements.

```
#include <stdio.h>
int main() {
   int day;
   printf("Enter a number (1-7): ");
   scanf("%d", &day);
      switch(day) {
      case 1:
         printf("Monday\n");
         break;
      case 2:
         printf("Tuesday\n");
         break;
      case 3:
         printf("Wednesday\n");
         break;
      case 4:
         printf("Thursday\n");
         break;
      case 5:
         printf("Friday\n");
         break;
      case 6:
         printf("Saturday\n");
         break;
      case 7:
         printf("Sunday\n");
         break;
      Default:
         print("Invalid day\n");
   }
   return 0;
}
```

Task 2: The following C program is to create a simple calculator using switch statements.

```
#include <stdio.h>
int main() {
    char operator;
    double num1, num2, result;
    printf("Enter an operator (+, -, *, /): ");
    scanf("%c", operator);
    printf("Enter two operands: ");
    scanf("%lf,%lf", &num1, num2);
    switch(operator) {
        case '+':
            result =num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
            result = num1 * num2;
            break;
        case '/':
            if(num2 != 0)
                result = num1 / num2;
            else
                printf("Division by zero error\n");
                result = 0;
            break;
        default
            printf("Invalid operator\n");
            result = 0;
            break;
}
```

```
    printf("Result: %lf\n", result);
    return 0;
}
```

Task 3: A C program to evaluate a student's grade based on their score using if-else statements

```
#include <stdio.h>
int main() {
    int score;
    char grade;
    printf("Enter the score: ");
    scanf("%d", score);
    if(score >= 90)
        grade = 'A';
    else if(score >= 80)
        grade = 'B';
    else if(score >= 70)
        grade = 'C';
    else if(score >= 60);
        grade = 'D';
    else
        grade = 'F';
    printf("The grade is: %d \n", grade);
    return 0;
}
```

Task 4: A C program to determine the largest of three numbers using if-else statements.

```
#include <stdio.h>
int main() {
    int a, b, c;
    printf("Enter three integers: ");
    scanf("%d %d %d", &a, &b, c);
    if(a >= b && a >= c) {
        printf("%d is the largest number\n", a);
    } else if(b >= a & b >= c) {
        printf("%d is the largest number\n", b);
    } else
        printf("%d is the largest number\n", c);
    }
    return 0;
}
```

LAB NO.: 3

LOOPING CONTROL STRUCTURES-WHILE & DO LOOPS

Objectives:

In this lab, student will be able to:

2. Write and execute C programs using ‘while’ statement
3. Write and execute C programs using ‘do-while’ statement
4. To learn to use break and continue statements in while and do while loop statements.

Introduction to while and do-while loops:

- 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);
```

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]

<u>Using ***do-while***</u>

```
#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;}
```

<u>Using ***while***</u>

```
#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 Exercise

Write C programs to do the following with the help of two iterative (looping) control structures: ***while*** and ***do-while*** statements

1. Reverse a given number and check if it is a palindrome or not. (use while loop).
   [Ex: 1234, reverse=$4*10^3 + 3 * 10^2 + 2 * 10^1 + 1 * 10^0$ =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. 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 Exercise

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. Write a program to demonstrate use of break and continue statements in while and do-while loops.

Debugging exercise

Instructions: Identify and correct the syntax or logical errors in the code.

Task 1: A C program to sum the first n natural numbers using a while loop.

```
#include <stdio.h>
int main() {
    int n, i = 1, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", n);
    while(i < n) {
        sum = sum + i;
        i++
    }
    printf("Sum of the first %d natural numbers is: %d\n", n, sum);
    return 0;
}
```

Task 2: A C program to calculate the factorial of a given number using a do-while loop

```
#include <stdio.h>
int main() {
    int n,num, i = 1;
    unsigned long long factorial = 1;
    printf("Enter an integer: ");
    scanf("%d", &num);
    do {
        factorial *= i;
        i++;
    } while(i <= n);
    printf("Factorial of %d = %llu\n", num, factorial);
    return 0;
}
```

Task 3: Write a C program to reverse the digits of a given number using a while loop.

```
#include <stdio.h>
int main() {
    int num, reversed = 0;
    printf("Enter an integer: ";
    scanf("%d", num);
```

```
        while(num != 0) ;{
        int digit = num % 10;
        reversed = reversed * 10 + digit;
        num /= 10;
    }
    printf("Reversed number is: %d\n", reversed);
    return 0;
}
```

LAB NO.: 4

LOOPING CONTROL STRUCTURES- FOR LOOPS

Objectives:

In this lab, student will be able to:

- Write and execute C programs using ‘for’ statement
- To learn to use break and continue statements in for loop statements.

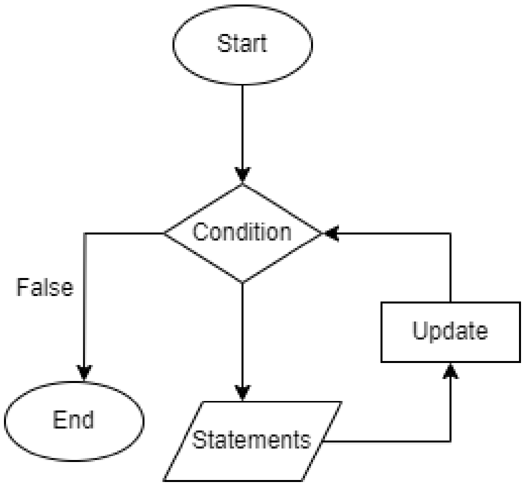
Introduction:

- For loop statements are used to repeat certain statements for a specified number of times.
- The statements are executed as long as the execution condition is true
- These types of control structures are also called as loop control structures

For loop:

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

For loop:



Lab Exercise

With the help of *for loop* statements, write C programs to do the following

1. 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  10  ….2*k
                        ..………………..…
                        n……………… n*k ]
```

2. 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
```

3. Evaluate the sine series, sin(x)= x- x^3/3! + x^5/5!–x^7/7!+ ……… to n terms.
4. 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

Additional Exercise

1. 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
2. Write a program to demonstrate use of break and continue statements in ***for*** loop.

Debugging exercise

Instructions: Identify and correct the syntax or logical errors in the code.

Task 1: A C program to calculate the sum of even numbers from 1 to n using a for loop.

```
#include <stdio.h>
int main() {
    int n, sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", n);

    for(int i = 1; i <= n; i++) {
        if(i % 2 = 0) {
            sum = sum + i;
        }
    }
    printf("Sum of even numbers from 1 to %d is: %d\n", n, sum);
    return 0;
}
```

Task 2: Write a C program to find and print all prime numbers up to a given number n using a for loop.

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);

    printf("Prime numbers up to %d are: ", n);

    for(int i = 2; i <= n; i++) {
        int isPrime = 1;
        for(int j = 2; j <= i/2 , j++) {
            if(i % j == 0) {
                isPrime == 0;
                break;
            }
        }
        if(isPrime) {
            printf("%d ", i);
        }
    }
    printf("\n");
    return 0;
}
```

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* is number of elements in 1-D array

data_type□is 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 Exercise

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. 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].
3. Print all the prime numbers in a given 1D array.
4. 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.

Cumulative Lab Programs: Towards Application Development

5. Define a 1D array called STUDENTS of size N, which holds the roll numbers of N students, display the roll numbers, search for a given roll number and display appropriate message.

STUDENTS

1	2	3	4	5	6	…	N

6. Define a 1D array called MARKS of size N, that holds the marks of N students, sort the marks in descending order and display the resultant array.

MARKS

23	79	68	93	37	89	…	N

Additional Exercise

Write C programs to do the following:

1. Insert an element into a 1D array, by getting an element and the position from the user.
2. Search an element in a 1D array using linear search.
3. Delete all the occurrences of the element present in the array which is inputted by the user.
4. To enter number of digits and create a number using this digit.

   [Hint: 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 Exercise

With the knowledge of 2D arrays, write C programs to do the following:

1. Display whether a given matrix is symmetric or not. [Hint: $A = A^T$]
2. Compute and display 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 and display the resultant matrix.
4. Interchange any two Rows & Columns in the given matrix and display resultant.

Cumulative Lab Programs: Towards Application Development

5. Write a C program to create a 2D array called STUDENTS_MARK of size 2xN to store the roll numbers of N students and corresponding marks scored by them and display STUDENTS_MARK.

   (Note: Create two 1D arrays STUDENTS and MARKS and assign roll numbers and marks respectively like the ones created in Lab 5, assign the contents of the two 1D arrays into the 2D array STUDENTS_MARK)

   Expected Output:
   ```
   1     2     3     4     5     6     …     N
   23    79    68    93    37    89    …     M1
   ```
6. Add the following to Q No. 5. Define another 2D array called STUDENTS_MARKS of size 4XN to store the roll numbers of N students along with the marks scored by them in three subjects respectively and display the 2D array.

(Note: Make use of the contents from the array STUDENTS_MARK in question no.5)

Expected Output:

```
1     2     3     4     5     6     …     N
23    79    68    93    37    89    …     M1
45    80    55    90    46    91    …     M2
39    88    65    89    50    70    …     M3
```

Additional Exercise

1. To interchange the primary and secondary diagonal elements in the given Matrix.
2. Search for an element in a given matrix and count the number of its occurrences.
3. Compute the row sum and column sum of a given matrix.
4. Check whether the given matrix is magic square or not.
5. Check whether the given matrix is a Lower triangular matrix or not.

```
Ex:    1  0  0
       2  3  0
       4  5  6
```

STRINGS

LAB NO.: 7

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 Exercise

With the brief introduction and knowledge on strings, write Cprograms 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. Delete a word from the given sentence.

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

   OUTPUT: I AM IN MIT

Cumulative Lab Programs: Towards Application Development

5. Add the following to Lab 6 program:

   Define a string array called STUDENTS_NAMES to store the names of N students and display them.

STUDENTS_NAMES

T	a	n	v	i	\0		
B	h	a	g	y	a	\0	
Y	a	s	h	\0			
A	b	h	a	y	\0		
. . .							
N	i	t	h	i	n	\0	

Refer the below expected output (>>>) samples for understanding:

STUDENTS_NAMES[0]   >>> “Tanvi”
STUDENTS_NAMES[0][3] >>> ‘v’

6. Append code to arrange the names of STUDENTS_NAMES (Refer Q. No.5) in alphabetical order and store it into another array of same dimension named STUDENTS_NAMES_SORTED and display sorted names. (Hint: use string handling function-***strcpy()***).

STUDENTS_NAMES_SORTED

A	b	h	a	y	\0		
B	h	a	g	y	a	\0	
N	i	t	h	i	n	\0	
T	a	n	v	i	\0		
. . .							
Y	a	s	h	\0			

Additional Exercise

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 ‘*’.
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 – FUNCTIONS

Objectives:

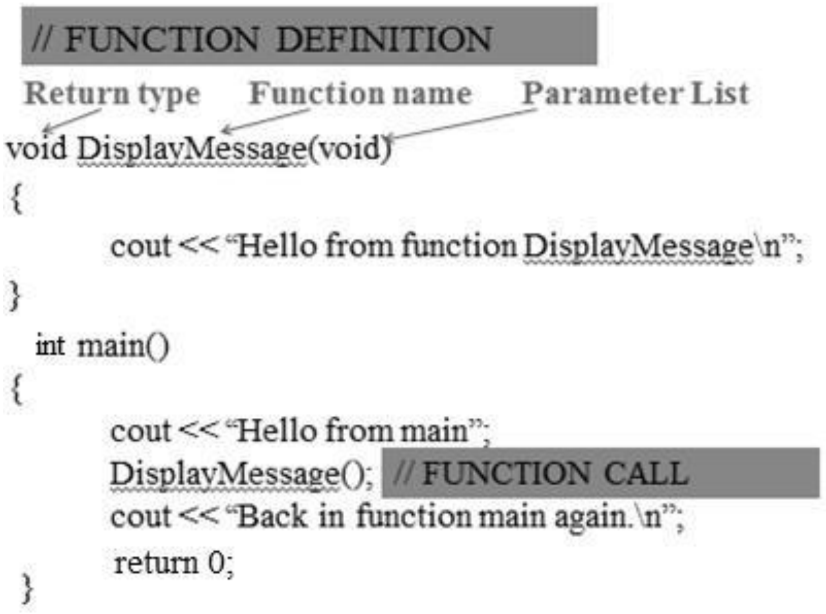
In this lab, student will be able to:

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. Learn the concept of recursion and implement recursive programs

Introduction – Modular Programming (Functions)

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

Recursive Function:

- 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

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 Exercise

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

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 **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 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.

4. Write a recursive function **FIB** to generate nth 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 ...]

Cumulative Lab Programs: Towards Application Development

5. Append C code to define a function called *Calculate_Average* which takes the array STUDENTS_MARKS, the number of students N (both from Lab 6) and a 1D array called AVERAGE of size N as arguments and computes the average marks of each student and stores it in the corresponding index of the array AVERAGE. Call this function in the *main()* and print the average marks of each student.

STUDENTS_MARK

1	2	3	4	5	6	…	N
23	79	68	93	37	89	…	M1
45	80	55	90	46	91	…	M2
39	88	65	89	50	70	…	M3

AVERAGE

1	2	3	4	5	6	…	N
35	82	62	90	44	83	…	M

6. Append 'C' code to define a function called Find_Topper which takes the array AVERAGE, the number of students N along with the array STUDENTS_NAMES (from Lab 7) as arguments and finds the student who has the maximum marks and prints the name of that student. Call this function in main().

Ex:

```
…
printf("The topper is : ");
Find_Topper(AVERAGE, N, STUDENTS_NAMES);
…
```

Expected Sample Output: The topper is : Abhay

Additional Exercise

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 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.
3. 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.
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.
5. Write a function **IsPalin** to check whether the given string is a palindrome or not. Write a main function to test this function.
6. Write a program to multiply two numbers using a recursive function. [Hint: Multiplication using repeated addition]

LAB NO.: 9

STRUCTERS AND POINTERS

Objectives:

In this lab, the student will be able to:

1. Learn the concept of structures and implement structure programs
2. Write basic operations and programs using structures
3. To learn the concept of pointers and solve various problems related to pointers
4. Declare and initialize pointer variable
5. Access a variable through its pointer

Introduction - Structures:

- The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type.
- The struct keyword is used to define the structure in the C programming language.
- The items in the structure are called its member and they can be of any valid data type.

C Structure Declaration

We have to declare structure in C before using it in our program. In structure declaration, we specify its member variables along with their datatype. We can use the struct keyword to declare the structure in C using the following syntax:

```
struct structure_name {
   data_type member_name1;
   data_type member_name1;
   ....
   ....
};
```

The above syntax is also called a structure template or structure prototype and no memory is allocated to the structure in the declaration.

C Structure Definition

To use structure in our program, we have to define its instance. We can do that by creating variables of the structure type. We can define structure variables using two methods

1. Structure Variable Declaration with Structure Template

```
struct structure_name {
   data_type member_name1;
   data_type member_name1;
   ....
   ....
}variable1, variable2, ...;
```

2. Structure Variable Declaration after Structure Template

```
// structure declared beforehand
struct structure_name variable1, variable2,.......;
```

Access Structure Members

We can access structure members by using the (.) dot operator.

```
structure_name.member1;
strcuture_name.member2;
```

A sample C program to demonstrate the working of structures

```
#include <stdio.h>
#include <string.h>
// create struct with person1 variable
struct Person {
  char name[50];
  int citNo;
  float salary;
} person1;
```

```
int main() {

  // assign value to name of person1
  strcpy(person1.name, "George Orwell");

  // assign values to other person1 variables
  person1.citNo = 1984;
  person1. salary = 2500;

  // print struct variables
  printf("Name: %s\n", person1.name);
  printf("Citizenship No.: %d\n", person1.citNo);
  printf("Salary: %.2f", person1.salary);

  return 0;
}
```

Output:

```
Name: George Orwell
Citizenship No.: 1984
Salary: 2500.00
```

typedef for Structures

The typedef keyword is used to define an alias for the already existing datatype. In structures, we have to use the struct keyword along with the structure name to define the variables. Sometimes, this increases the length and complexity of the code. We can use the typedef to define some new shorter name for the structure.

A sample C program to demonstrate the working of typedef in structures

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

```
#include <string.h>

// struct with typedef person
typedef struct Person {
  char name[50];
  int citNo;
  float salary;
} person;  // Here, we have used typedef with the Person structure to create an alias person.

int main() {

  // create  Person variable
  person p1;  //  Now, we can simply declare a Person variable using the person alias:

  // assign value to name of p1
  strcpy(p1.name, "George Orwell");

  // assign values to other p1 variables
  p1.citNo = 1984;
  p1. salary = 2500;

  // print struct variables
  printf("Name: %s\n", p1.name);
  printf("Citizenship No.: %d\n", p1.citNo);
  printf("Salary: %.2f", p1.salary);

  return 0;
}
```

Nested Structures
You can create structures within a structure in C programming.

A sample C program to demonstrate the working of nesting in structures

```
#include <stdio.h>
// child structure declaration
struct child {
    int x;
    char c;
};
  // parent structure declaration
struct parent {
    int a;
    struct child b;
};
int main()
{
    struct parent var1 = { 25, 195, 'A' };
    // accessing and printing nested members
    printf("var1.a = %d\n", var1.a);
    printf("var1.b.x = %d\n", var1.b.x);
    printf("var1.b.c = %c", var1.b.c);
    return 0;
}
```

Array of Structures
An array whose elements are of type structure is called array of structure. It is generally useful when we need multiple structure variables in our program.
A sample C program to demonstrate the working of arrays and structures

```
// C program to demonstrate the array of structures
#include <stdio.h>
// structure template
 struct Employee {
   char Name[20];
    int employeeID;
    int WeekAttendence[7];
};
```

```
 int main()
{
   // defining array of structure of type Employee
   struct Employee emp[5];
   // adding data
   for (int i = 0; i < 5; i++) {
      emp[i].employeeID = i;
      strcpy(emp[i].Name, "Amit");
      int week;
      for (week = 0; week < 7; week++) {
         int attendence;
         emp[i].WeekAttendence[week] = week;
      }
   }
   printf("\n");
   // printing data
   for (int i = 0; i < 5; i++) {
      printf("Emplyee ID: %d - Employee Name: %s\n",
            emp[i].employeeID, emp[i].Name);
      printf("Attendence\n");
      int week;
      for (week = 0; week < 7; week++) {
         printf("%d ", emp[i].WeekAttendence[week]);
      }
      printf("\n");
   }
   return 0;
}
```

Introduction to Pointers:

- A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

- As the pointers in C store the memory addresses, their size is independent of the type of data they are pointing to. This size of pointers in C only depends on the system architecture.

Syntax of C pointers

The syntax of pointers is similar to the variable declaration in C, but we use the (* dereferencing operator in the pointer declaration.

datatype * ptr;

where

ptr is the name of the pointer.

datatype is the type of data it is pointing to

How to Use Pointers?

The use of pointers in C can be divided into three steps:

1. **Pointer Declaration**
2. **Pointer Initialization**
3. **Pointer Dereferencing**

1. Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.

```
int *ptr;
```

2. Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&) addressof operator** to get the memory address of a variable and then store it in the pointer variable.

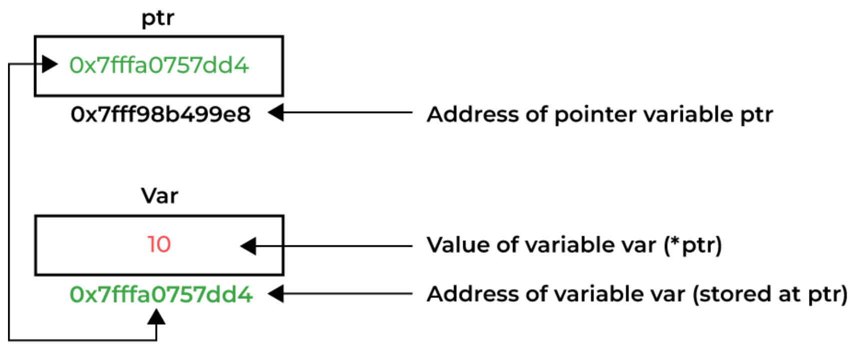
```
int var = 10;
int * ptr;
ptr = &var;
```

We can also declare and initialize the pointer in a single step. This method is called **pointer definition** as the pointer is declared and initialized at the same time.

int ***ptr = &**var;

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same **(*) dereferencing operator** that we used in the pointer declaration.

Dereferencing pointers in C


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (stored at ptr)

A sample C program to demonstrate the working of pointers

```
void DemonstratePointers()
{
   int var = 10;
   // declare pointer variable
   int* ptr;
   // note that data type of ptr and var must be same
   ptr = &var;
   // assign the address of a variable to a pointer
   printf("Value at ptr = %p \n", ptr);
   printf("Value at var = %d \n", var);
   printf("Value at *ptr = %d \n", *ptr);
}
int main()
{
   DemonstratePointers();
   return 0;
}
```

Output:

```
Value at ptr = 0x7fff1038675c
Value at var = 10
Value at *ptr = 10
```

A few basic pointer types in C

Pointers in C can be classified into many different types based on the parameter on which we are defining their types. If we consider the type of variable stored in the memory location pointed by the pointer, then the pointers can be classified into the following types:

1. Integer Pointers: As the name suggests, these are the pointers that point to the integer values.

```
        int *ptr;
```

2. Array Pointer: Pointers and Array are closely related to each other. Even the array name is the pointer to its first element. They are also known as Pointer to Arrays. We can create a pointer to an array using the given syntax.

```
        char *ptr = &array_name;
```

3. Function Pointers: Function pointers point to the functions. They are different from the rest of the pointers in the sense that instead of pointing to the data, they point to the code. Let's consider a function prototype – int func (int, char), the function pointer for this function will be

```
        int (*ptr)(int, char);
```

A sample C program to demonstrate the working of increment in a pointer

```
#include <stdio.h>
int main() {
    // Declare an array
    int v[3] = { 10, 100, 200 };

    // Declare pointer variable
    int* ptr;
    // Assign the address of v[0] to ptr
    ptr = v;
    for (int i = 0; i < 3; i++) {
        // print value at address which is stored in ptr
        printf("Value of *ptr = %d\n", *ptr);

        // print value of ptr
        printf("Value of ptr = %p\n\n", ptr);
        // Increment pointer ptr by 1
        ptr++;
    }
    return 0;
}
```

Output:

```
Value of *ptr = 10
Value of ptr = 0x7ffe8ba7ec50

Value of *ptr = 100
Value of ptr = 0x7ffe8ba7ec54

Value of *ptr = 200
Value of ptr = 0x7ffe8ba7ec58
```

Structure Pointer in C

We can define a pointer that points to the structure like any other variable. Such pointers are generally called Structure Pointers. We can access the members of the structure pointed by the structure pointer using the (->) arrow operator.

A sample C program to demonstrate the working of pointers and structures

```
// C program to illustrate the structure pointer
#include <stdio.h>

// structure declaration
struct Point {
   int x, y;
};

int main()
{
   struct Point str = { 1, 2 };

   // p2 is a pointer to structure p1
   struct Point* ptr = &str;

   // Accessing structure members using structure pointer
   printf("%d %d", ptr->x, ptr->y);

   return 0;
}
```

Output

```
1 2
```

Lab Exercise

With knowledge of structures and pointers

Write C programs as specified below:

1. Find the maximum number in the input integer array using pointers.
2. Write a C program to swap two numbers using pointers (Call by reference)
3. Define a structure to store student details (e.g., name, roll number, marks in three subjects). Write a program to input data for five students, calculate their total and average marks, and display the results.

4. Define a structure for a student with a nested structure for the address (including street, city, and zip code). Write a program to input and display student details along with their address (Use nested structure concept)

5. Define a structure for a product with members for product ID, name, and price. Write a program to create array of products using pointers, input their details, and display them. (Use the concept of pointers and structures)

6. Define a structure for a cricket player with members for player name, team name, and batting average. Write a program to input data for multiple players and sort them by batting average. (Use the concept of array and structures)

Additional Exercise

1. Write C programs to perform the following pointer arithmetic operations (a) Decrement in a Pointer (b) Addition of integer to a pointer (c) Subtraction of integer to a pointer.

2. Write a C program to copy one array to another using pointers

3. Define a structure called Date with members for day, month, and year. Write a program to read two dates and determine which one is earlier.
4. Define a structure for a book with members for title, author, ISBN, and price. Write a program to manage a collection of books, including adding, deleting, and searching for books by title or author.

LAB NO.: 10

‘C’ FILE HANDLING

Objectives:
In this lab, student will be able to:

1. Distinguish between console oriented and file oriented i/o.
2. Open and write to a text file using C.
3. Open and read from a text file using C.
4. Errors and Error Handling

File – place on disc where group of related data is stored.

- Example: C programs, executable files, etc.

C supports a number of functions that have the ability to perform basic file operations, which include:

- Naming
- Opening
- Reading
- Writing
- Closing

Defining and opening file

- To store data file in secondary memory (disc) it must specify to OS:
    - **Filename** (e.g. sort.c, input.data)
    - **Data structure** (e.g. FILE)
    - **Purpose** (e.g. reading, writing, appending)
- String of characters that make up a valid filename for OS may contain two parts
    - **Primary** (name)
    - Optional **period** with **extension**
    - Examples: **a.txt**, **program.c**, **temp**, **text.out**

The general format of the function used for opening a file is

```
FILE *fp;
fp=fopen(“filename”,”mode”);
```

The first statement declares the variable fp as a pointer to the data type FILE. As stated earlier, File is a structure that is defined in the I/O Library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. This pointer, which contains all the information about the file, is subsequently used as a communication link between the system and the program.

The second statement also specifies the purpose of opening the file. The mode does this job. Mode can be

r - open the file for read only.
w - open the file for writing only.
a - open the file for appending data to it.

Consider the following statements:

```
FILE *p1, *p2;
p1=fopen(“data”,”r”);
p2=fopen(“results”,”w”);
```

In these statements the p1 and p2 are created and assigned to open the files data and results respectively, the file data is opened for reading and result is opened for writing. In case the results file already exists, its contents are deleted and the files are opened as a new file. If data file does not exist error will occur.

Lab Exercise:
With knowledge of file handling concepts
Write C programs for the following

1. To open and read a sentence from a file and display the same on the console.
2. To write a line of text into an existing file.
3. To copy the contents of one file into another file.
4. To read a file and write into another file converting all characters to upper case.
5. To count and display the number of characters, words and lines of a file.
6. To print the last n characters of a file. Input ‘file name’ and ‘n’ value from console.

LAB NO.: 11

INTRODUCTION TO GIT AND GITHUB

Introduction To Git

Git is a distributed version control system designed to track changes in source code during software development. It is the open source version control system started by Linus Trovalds. It's primary purpose is to manage projects with speed and efficiency, allowing multiple developers to work on a codebase simultaneously without overwriting each other's contributions. It allows each developer to have a complete copy of the project repository, facilitating offline work and enhancing redundancy. This decentralized approach adopted in Git allows for collaboration and ensures data integrity. The powerful branching and merging capabilities, making Git a popular choice among software development teams.

Installation of Git

The steps required to install Git in Ubuntu operating system are as follows:

- Update the local package index using the apt package management tools
  sudo apt update
- Enter the following command in the terminal to install Git
  sudo apt install git
- Confirm the installation of Git by running the following commands
  git --version

Git Configuration

After successful installation of Git it must be configured so that the generated commit messages contains correct information and supports while building a software project. Configuration can be achieved by using the **git config** command.

- Enter the username and email address using following command
  git config --global user.name "your name"
  git config --global user.email "youremail@domain.com"
- Display the configured items by typing
  git config --list

Basic Git Commands

- **git init -** It initializes the current folder as the git repository
- **git status** – It displays the current status of the folder
- **git add** – It adds the files to the git repository.
- **git add .** - It adds all files to the git repository.
- **git commit** – It commits the changes to the git repository.
- **git push** – It uploads local repository content to a remote repository.
- **git pull** – It fetches and downloads content from a remote repository and immediately update the local repository to match that content.

Introduction To GitHub

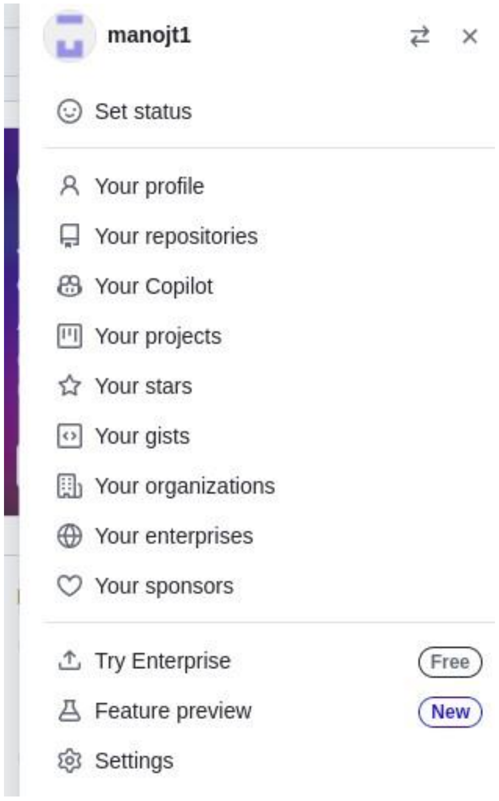
GitHub is a web-based platform that uses Git, the distributed version control system, to facilitate collaboration and version control for software development projects. Founded in 2008 and now owned by Microsoft, GitHub provides a user-friendly interface for managing Git repositories, enabling developers to store, share, and collaborate on code more effectively. Its primary purpose is to streamline the development process by offering tools for issue tracking, code review, project management, and continuous integration/continuous deployment. By hosting repositories in the cloud, GitHub allows developers from around the world to contribute to projects, track changes, and work together seamlessly, making it an essential tool in the modern software development ecosystem.

Creating a GitHub Account

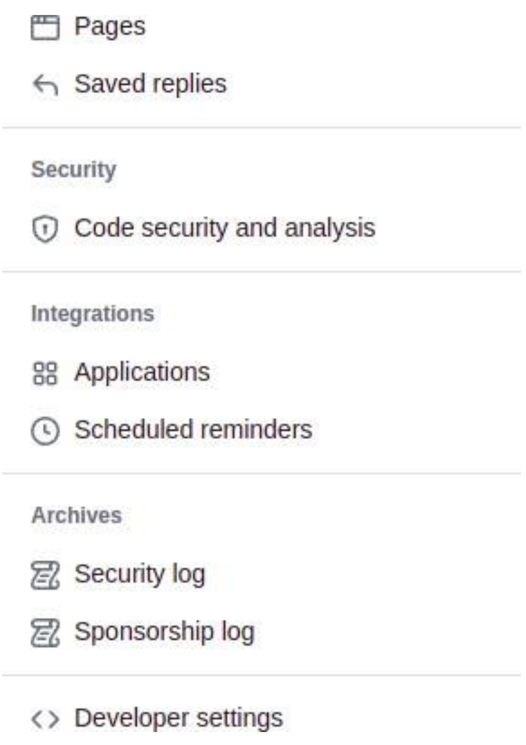
The steps to be followed to create an GitHub account is as follows:

- Browse to GitHub's homepage: https://github.com/
- At the top right, click on the **Sign up** button.
- Enter an email address (preferably your learner id email), choose a unique and secure password. Enter an original and professional username and choose whether you would like to receive marketing emails. Solve the **"verify your account"** puzzle. Then click on the **Create account** button.
- GitHub will send a verification email with a code to enter in the web browser.
- Answer the questions that GitHub asks about the programming experience and goals, and choose **"Continue for free"** on the pricing page.
- Eventually, GitHub displays the new dashboard page, with options to create a new repository, set up the profile, etc.

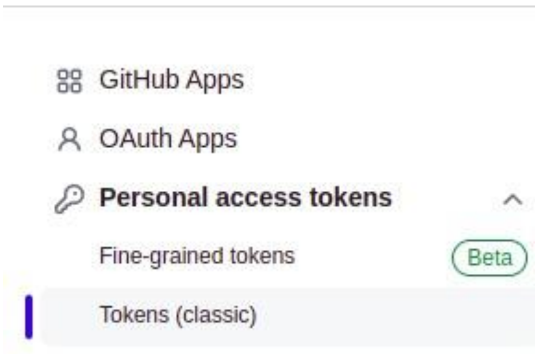
- Set up a personal access token. GitHub requires the personal access token instead of your account password when operating from the command line.
- The instructions for creating a personal access token are detailed here:
  - ➢ From your profile icon in the upper right corner select **Settings** .
  - ➢ On the page that loads select **Developer Settings** from the bottom of the menu on the left.
  - ➢ On the page that loads select **Personal access tokens** and choose **Tokens (classic)**; then click **Generate new token (classic)**.
  - ➢ For the configuration options: check the **repo** checkbox to give the token permissions to access your repositories, and choose an approriate expiration date.
  - ➢ It is important to remember the token – you need to copy it to a file on the computer. You will use this each time you push to or pull from your GitHub repository.

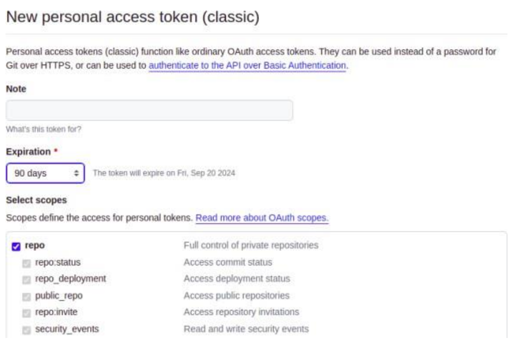


Select Settings



Go to Developer Settings





New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

What's this token for?

Expiration *

90 days — The token will expire on Fri, Sep 20 2024

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ **repo**	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events

Select Personal Access Tokens

Generate Token

Exercise 1 : Creating a GitHub Repository and Adding Files

- Click on the **Create a repository** button.
- In the resulting page, for **Repository name** enter **MyFirstCProject**, select the **Private radio button**.
- Check the initialize this repository with a **README** checkbox, and click on the **Create repository** button.
- Open a **terminal window** on the computer. Change the working directory to the one where you want your development Git repository to reside using **cd** command.
- Now, issue the following command

  git clone https://github.com/yourgithubusername/MyFirstCProject

  For a particular user the above command looks like : git clone https://github.com/manojt1/MyFirstCProject

  When prompted, enter your GitHub username and GitHub personal access token. Now, the working directory contains a new directory named **MyFirstCProject** which contains a **README.md** file and a **.git** directory.
- To add new files to the local repository in the terminal window change the working directory to your Git repository using cd command and pull any updates from GitHub repository

  git pull
- Add the files using following command[Create the required files before adding them]

```
git add filename1.c filename2.c
```

- Commit the staged files to the local repository

```
git commit -m "Message describing the commit"
```

- Push the updated local repository to the GitHub repository **MyFirstCProject** created earlier

```
git push origin main
```

- Check the status with following command

```
git status
```

Exercise 2 : Push an Existing Project to GitHub

- From the computer terminal create a working directory for project named **TicTacToeProject** using following command

```
git init home/username/TicTacToeProject
```

- Create the necessary files related to project and add it to local Git repository

```
git add tictactoe_main.c tictactoe_util.c tictactoe_util.h
```

- Commit the staged files to the local Git repository using following command

```
git commit -m "Adding main and util files"
```

- Now Sign in to your GitHub account and create remote repository named **TicTacToeProject** with **Private** access and **README** file.
- Connect the local repository to the remote and verify it using following commands

```
git remote add origin https://github.com/username/TicTacToeProject
git remote -v
```

- Push the updates in the local repository to remote using following command

```
git push -u -f origin main
```
