

**LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING  
(AUTONOMOUS)**

**Programming for Problem Solving using C Lab Manual  
I Year I Semester (R20)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## Vision of the Department

The Computer Science & Engineering aims at providing continuously stimulating educational environment to its students for attaining their professional goals and meet the global challenges.

## Mission of the Department

- **DM1:** To develop a strong theoretical and practical background across the computer science discipline with an emphasis on problem solving.
- **DM2:** To inculcate professional behaviour with strong ethical values, leadership qualities, innovative thinking and analytical abilities into the student.
- **DM3:** Expose the students to cutting edge technologies which enhance their employability and knowledge.
- **DM4:** Facilitate the faculty to keep track of latest developments in their research areas and encourage the faculty to foster the healthy interaction with industry.

## Program Educational Objectives (PEOs)

- **PEO1:** Pursue higher education, entrepreneurship and research to compete at global level.
- **PEO2:** Design and develop products innovatively in computer science and engineering and in other allied fields.
- **PEO3:** Function effectively as individuals and as members of a team in the conduct of interdisciplinary projects; and even at all the levels with ethics and necessary attitude.
- **PEO4:** Serve ever-changing needs of society with a pragmatic perception.

## PROGRAMME OUTCOMES (POs):

<b>PO 1</b>	<b>Engineering knowledge:</b> Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
<b>PO 2</b>	<b>Problem analysis:</b> Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
<b>PO 3</b>	<b>Design/development of solutions:</b> Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
<b>PO 4</b>	<b>Conduct investigations of complex problems:</b> Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
<b>PO 5</b>	<b>Modern tool usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
<b>PO 6</b>	<b>The engineer and society:</b> Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
<b>PO 7</b>	<b>Environment and sustainability:</b> Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
<b>PO 8</b>	<b>Ethics:</b> Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
<b>PO 9</b>	<b>Individual and team work:</b> Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
<b>PO 10</b>	<b>Communication:</b> Communicate effectively on complex engineering activities with the

	engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
<b>PO 11</b>	<b>Project management and finance:</b> Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
<b>PO 12</b>	<b>Life-long learning:</b> Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

#### **PROGRAMME SPECIFIC OUTCOMES (PSOs):**

<b>PSO 1</b>	The ability to apply Software Engineering practices and strategies in software project development using open-source programming environment for the success of organization.
<b>PSO 2</b>	The ability to design and develop computer programs in networking, web applications and IoT as per the society needs.
<b>PSO 3</b>	To inculcate an ability to analyze, design and implement database applications.

## **Module 1: Introduction to Raptor Tool**



## Introduction to Algorithmic Thinking

### What is an algorithm?

An algorithm is a precise, step-by-step set of instructions for solving a task. An algorithm does not solve a task; it gives you a series of steps that, if executed correctly, will result in a solution to a task. You use algorithms every day, but you often do not explicitly think about the individual steps of the algorithm. For example, starting your car, putting on your clothes, logging into your computer, or following a recipe for cooking a food dish, are all accomplished using an algorithm, a step-by-step series of actions.

For an algorithm to be valid, each step (or instruction) must be:

- unambiguous – the instruction can only be interpreted in one unique way.
- executable – the person or device executing the instruction must know how to accomplish the instruction without any extra information.
- ordered – the steps of an algorithm must be ordered in a proper sequence to correctly accomplish the task.

An example algorithm is shown in the seven steps below. This algorithm explains how to make "frozen lemon ice-box pie".

Step 1: Pour 1 can of chilled pet milk into a mixing bowl.

Step 2: Beat the pet milk until peaks form.

Step 3: "Fold in" 1 cup of sugar and 1/2 cup of lemon juice.

Step 4: If you want a lemonier taste, add 1 tsp of grated lemon peel.

Step 5: Spread a graham cracker crust on the bottom of a 13" by 9" pan.

Step 6: Pour the pet milk mix into the pan.

Step 7: Place in freezer until frozen

### What is Algorithmic Thinking?

Algorithmic thinking is the ability to understand, execute, evaluate, and create algorithms. Let's discuss each of these ideas separately.

To be an algorithmic thinker you need the ability to understand and execute algorithms. Some people find it easy to follow a series of precise instructions while other people find it very challenging. Some people seem to lack the patience and diligence required to follow a step-by-step plan. However, it is a valuable skill that all people should master. Algorithmic thinking requires patience because each instruction must be executed in its correct sequence without skipping ahead or "glossing over" some of the instructions. In addition, algorithmic thinking requires diligence and perseverance. It is

often tedious to follow the steps of a complex algorithm and people sometimes fail to complete an algorithm because they simply "give up."

Algorithmic thinking also requires the ability to evaluate algorithms. This involves determining if an algorithm really does solve a given task. This can be very challenging. For example, a "pre-flight check list" is an algorithm for preparing an aircraft for take-off. Suppose you were given the job of determining if a new "pre-flight check list" for the F35 (Joint Strike Fighter) is correct (checks all systems on the aircraft properly) and complete (does not leave out any important checks). Hopefully you would agree that this is an important job – pilot's lives are dependent on getting it right – and that getting the check list correct and complete will not be easy. (As a side note, one way to create the pre-flight check list is to determine the fault of each new aircraft crash and add a new check to the list to prevent future crashes.

And finally, Algorithmic thinking includes the ability to create new algorithms. This is probably the most challenging aspect of algorithmic thinking. Given a task, can you create a series of precise, step-by-step instructions that always solves the task correctly? Obviously, the complexity of the task has a big impact on the complexity of an algorithm that will accomplish the task.

A critical aspect of algorithm creation is the "target executer" of an algorithm. For example, the "target executer" of a pre-flight check list is a pilot. The average person on the street could not execute the pre-flight check list correctly because they would not understand the instructions. The three, previously stated, characteristics of an algorithm instruction (i.e., an instruction must be unambiguous, executable, and ordered) must be consistent with the knowledge and expertise of the "target executer." When you create an algorithm, make sure that each instruction can be unambiguously interpreted and executed by the "target executer."

### **Why are Algorithms Important?**

Algorithms are important for many reasons:

- An algorithm documents the "how to" for accomplishing a particular task.
- If an algorithm is written well, it can be used to accomplish not only a single task but a whole group of related tasks.
- The existence of an algorithm means that the task can potentially be automated (i.e., performed by a computer).
- The automation of redundant, tedious, or dangerous tasks frees people from having to perform these boring, time-consuming, or potentially lethal tasks.
- The automation of some tasks makes new things possible (e.g., accessing web pages from all over the entire world in the blink of an eye).

**What are the important properties of algorithms?**

The following is a list of some of the important properties of algorithms. This list is by no means comprehensive or complete.

**Property 1:** For any given, non-trivial task (or set of related tasks), there are many possible algorithms for accomplishing the task.

**Property 2:** An algorithm does not encode the underlying theory behind the instruction steps.

**Property 3:** Some algorithms are more efficient than other algorithms.

**Property 4:** Computer programs that are used over many years typically must be modified over time to adapt to changes in task requirements.

**How to create an algorithm?**

**Principle 1:** Solve a specific instance of the problem by hand using pencil and paper.

**Principle 2:** Generalize your solution by replacing your specific instance values with "variables".

**Principle 3:** Manually execute your algorithm on several test cases to verify that it produces correct answers. This is called desk checking or walking through the algorithm.

**Principle 4:** After a general solution to the task is known, write correct programming statements to implement your solution on the computer.

\*\*\*\*\*

## Introduction to RAPTOR Tool

### What is RAPTOR?

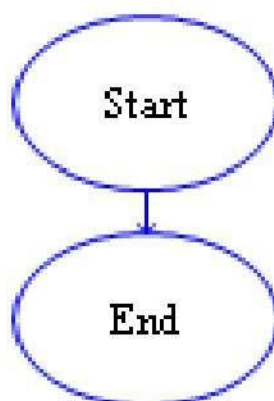
RAPTOR is a visual programming development environment based on flowcharts. A flowchart is a collection of connected graphic symbols, where each symbol represents a specific type of instruction to be executed. The connections between symbols determine the order in which instructions are executed. These ideas will become clearer as you use RAPTOR to solve problems.

We use RAPTOR for several reasons:

- The RAPTOR development environment minimizes the amount of syntax you must learn to write correct program instructions.
- The RAPTOR development environment is visual. RAPTOR programs are diagrams (directed graphs) that can be executed one symbol at a time. This will help you follow the flow of instruction execution in RAPTOR programs.
- RAPTOR is designed for ease of use. (You might have to take our word for this, but other programming development environments are extremely complex.)
- RAPTOR error messages are designed to be more readily understandable by beginning programmers.
- Our goal is to teach you how to design and execute algorithms. These objectives do not require a heavy-weight commercial programming language such as C++ or Java.

### RAPTOR Program Structure:

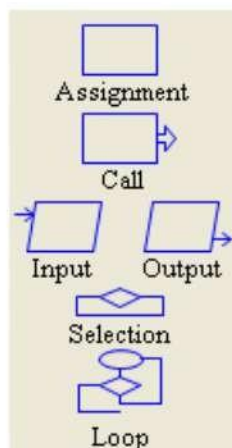
A RAPTOR program is a set of connected symbols that represent actions to be performed. The arrows that connect the symbols determine the order in which the actions are performed. When executing a RAPTOR program, you begin at the Start symbol and follow the arrows to execute the program. A RAPTOR program stops executing when the End symbol is reached. The smallest RAPTOR program (which does nothing) is depicted at the right. By placing additional RAPTOR statements between the Start and End symbols you can create meaningful RAPTOR programs.





### Introduction to RAPTOR Statements/Symbols:

RAPTOR has six (6) basic symbols, where each symbol represents a unique type of instruction. The basic symbols are shown at the right. The top four statement types, Assignment, Call, Input, and Output are explained in this reading, The bottom two types, Selection and Loops, will be explained in a future reading.



The typical computer program has three basic components:

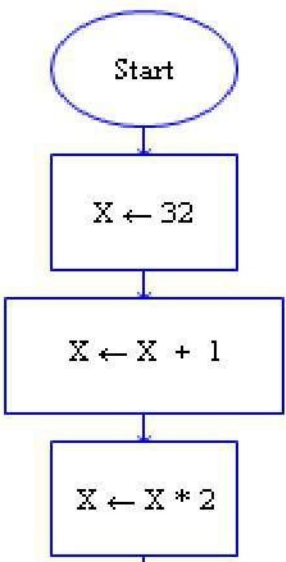
- INPUT – get the data values that are needed to accomplish the task.
- PROCESSING – manipulate the data values to accomplish the task.
- OUTPUT – display (or save) the values which provide a solution to the task.

These three components have a direct correlation to RAPTOR instructions as shown in the following table.

Purpose	Symbol	Name	Description
INPUT		input statement	Allow the user to enter data. Each data value is stored in a <b>variable</b> .
PROCESSING		assignment statement	Change the value of a <b>variable</b> using some type of mathematical calculation.
PROCESSING		procedure call	Execute a group of instructions defined in the named procedure. In some cases, some of the procedure arguments (i.e., <b>variables</b> ) will be changed by the procedure's instructions.
OUTPUT		output statement	Display (or save to a file) the value of a <b>variable</b> .

**RAPTOR Variables:**

Variables are computer memory locations that hold a data value. At any given time, a variable can only hold a single value. However, the value of a variable can vary (change) as a program executes. That's why we call them "variables"! As an example, study the following table that traces the value of a variable called X.

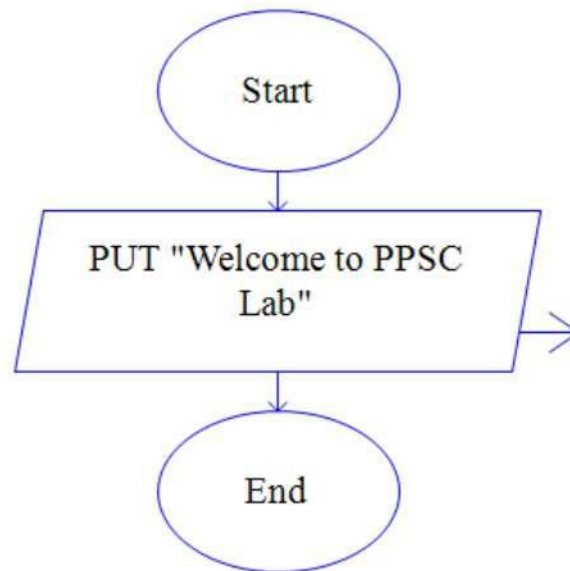
Description	Value of X	Program
When the program begins, no variables exist. In RAPTOR, variables are automatically created when they are first used in a statement	Undefined	 <pre> graph TD     Start([Start]) --&gt; A[X ← 32]     A --&gt; B[X ← X + 1]     B --&gt; C[X ← X * 2]     C --&gt; End[ ]   </pre>
The first assignment statement, $X \leftarrow 32$ , assigns the data value 32 to the variable X	32	
The next assignment statement, $X \leftarrow X + 1$ , retrieves the current value of X, 32, adds 1 to it, and puts the result, 33, in the variable X	33	
The next assignment statement, $X \leftarrow X * 2$ , retrieves the current value of X, 33, multiplies it by 2, and puts the result, 66, in the variable X	66	

\*\*\*\*\*

## **Module 2: Problem solving using Raptor Tool**

a) Design a Flow Chart using RAPTOR to print the following message: “Welcome to PPSC Lab”.

**Flow Chart:**

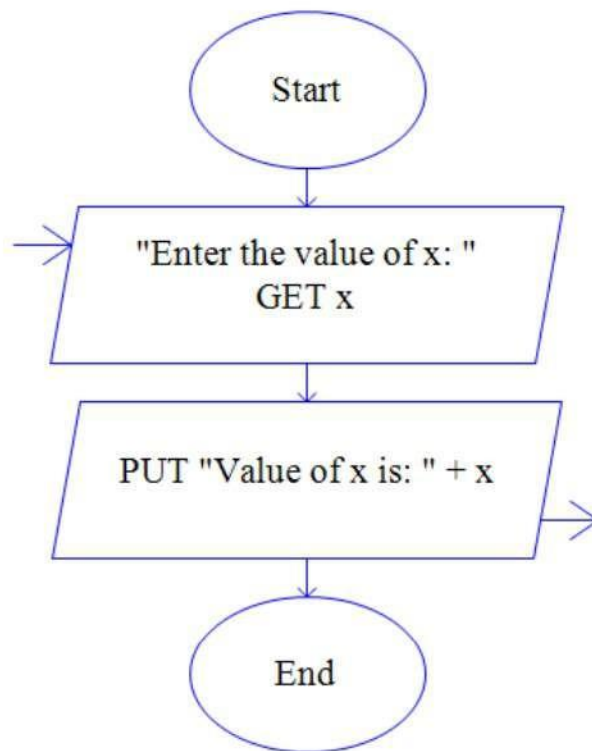


**Output:**



**b) Design a Flow Chart using RAPTOR for reading a Number and Printing that number.**

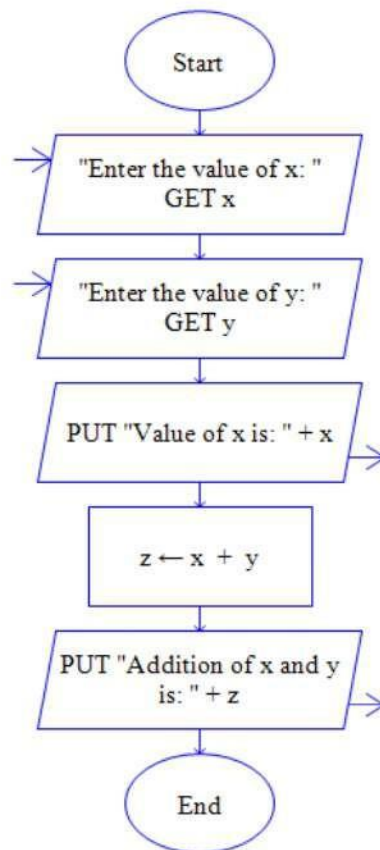
**Flow Chart:**



**Output:**

c) Design a Flow Chart using RAPTOR for performing Addition of TWO Numbers.

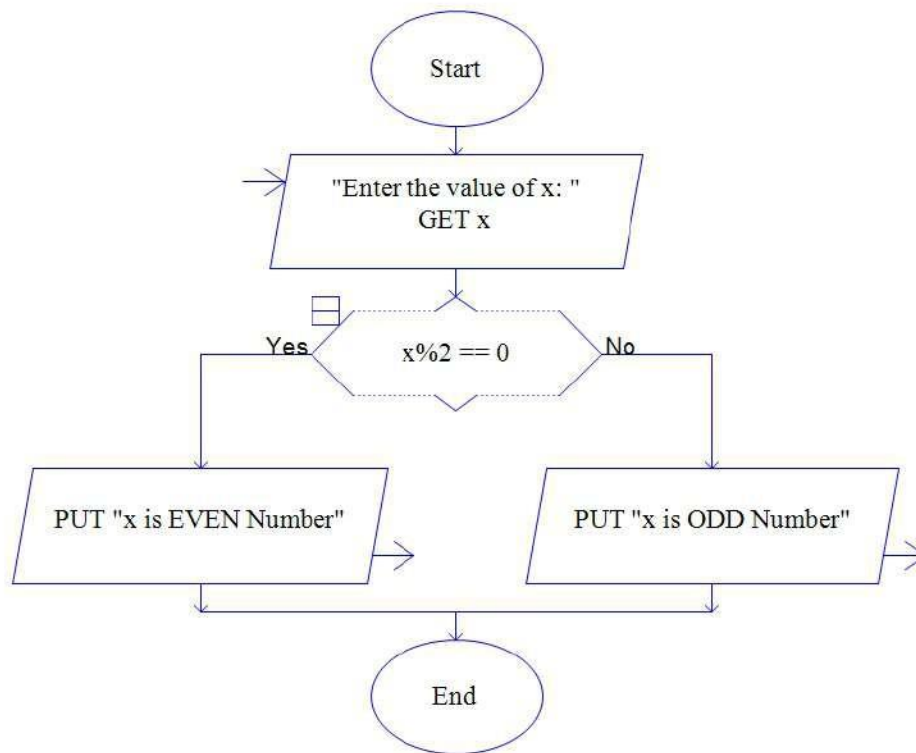
**Flow Chart:**



**Output:**

d) Design a Flow Chart using RAPTOR to check whether a number is EVEN or ODD.

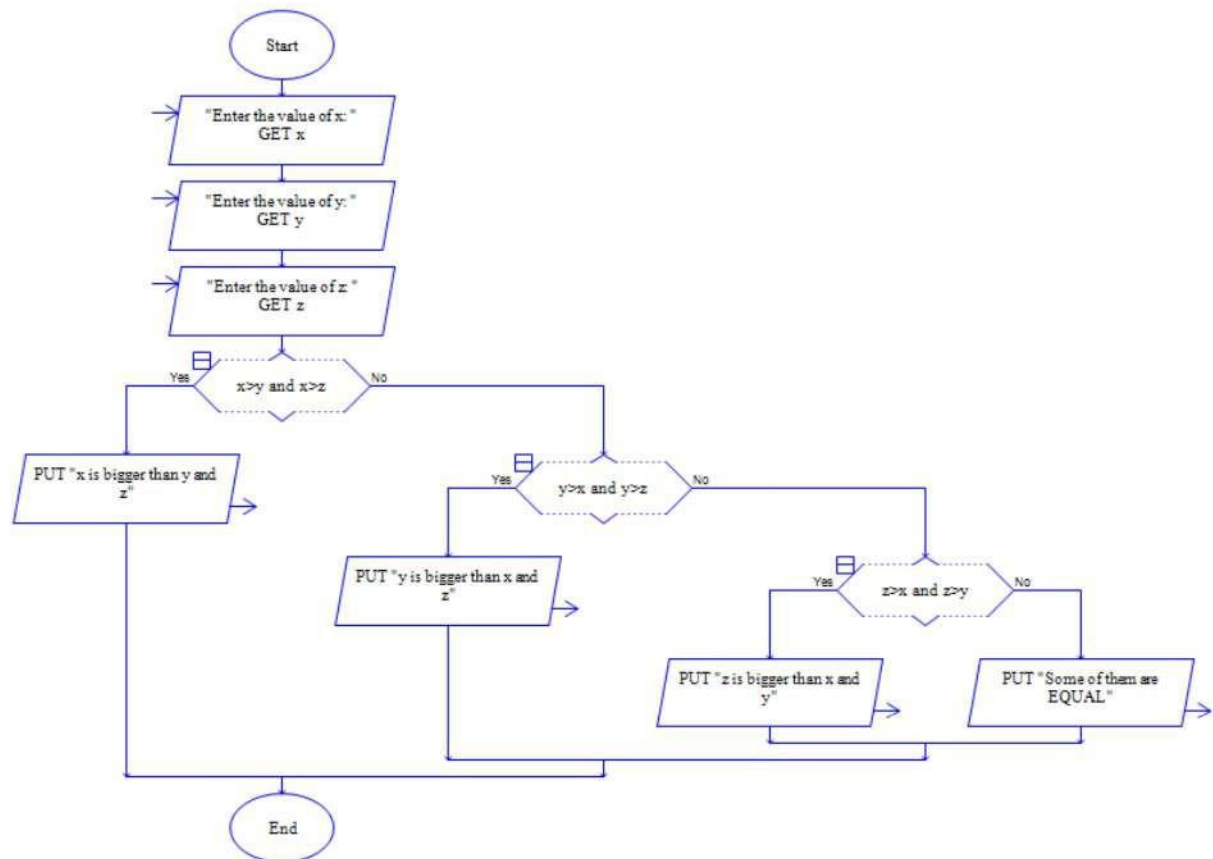
**Flow Chart:**



**Output:**

e) Design a Flow Chart using RAPTOR to check biggest among THREE Numbers.

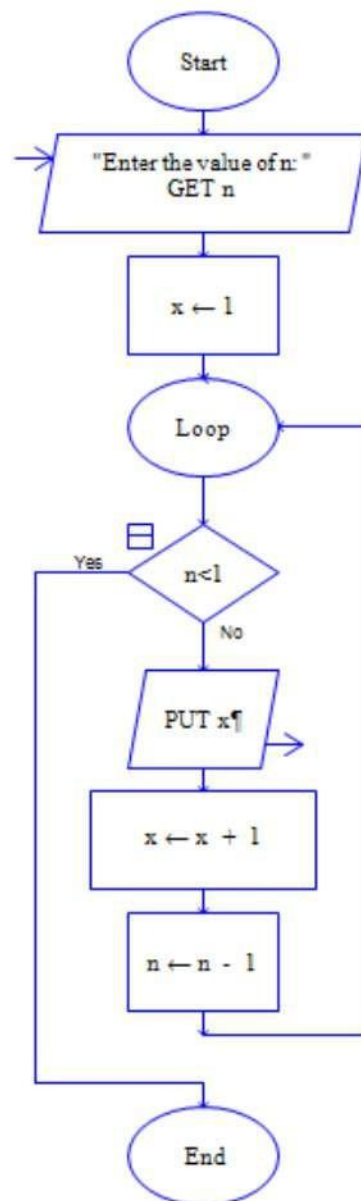
**Flow Chart:**



**Output:**

f) Design a Flow Chart using RAPTOR to print 1 to n Numbers.

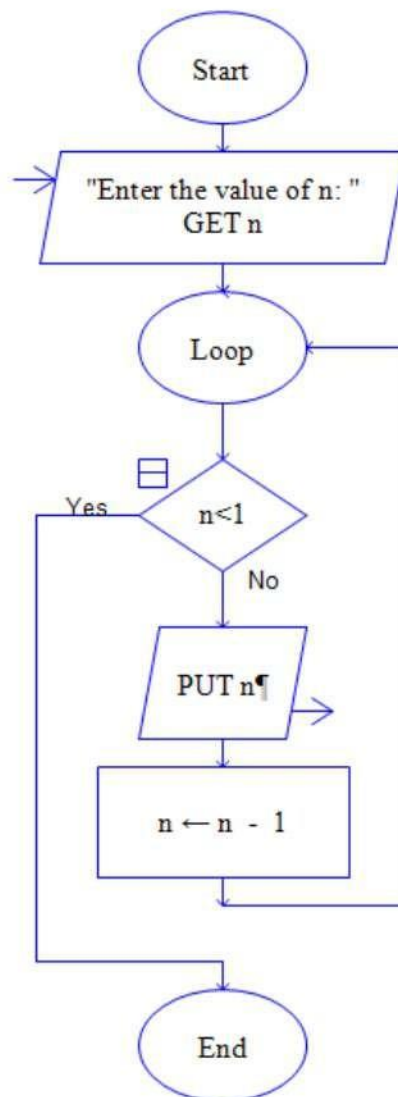
**Flow Chart:**



**Output:**

**g) Design a Flow Chart using RAPTOR to print n to 1 Numbers.**

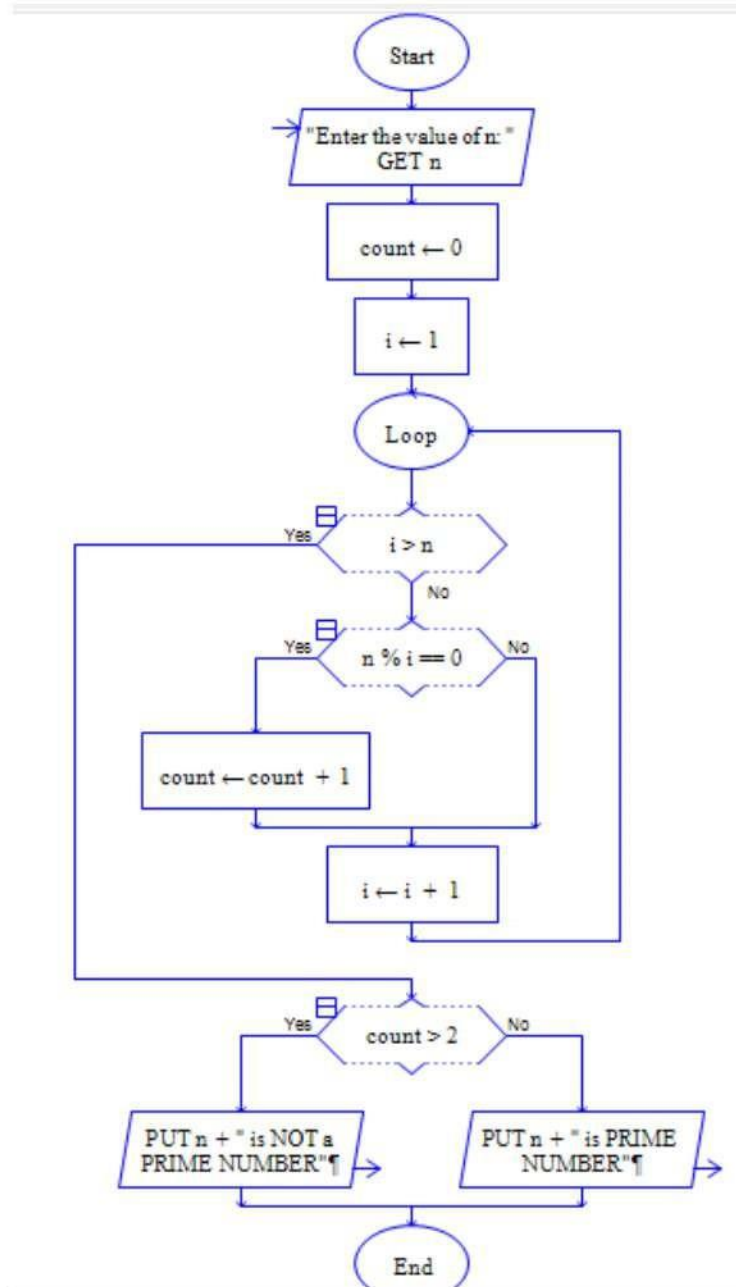
**Flow Chart:**



**Output:**

**h) Design a Flow Chart using RAPTOR to check whether a Number is Prime or Not.**

**Flow Chart:**



**Output:**

## **Module 3: Exercise Programs on Basics of C-Program**



**a) Write a Program to Print the following Message: “Hello Students”**

**Program:**

```
#include <stdio.h>

int main( )
{
    printf("Hello Students");

    return 0;
}
```

**Output:**

**b) Write a Program to read an Integer value from STDIN into variable and print that value.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int x;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Value of x is: %d", x);

    return 0;
}
```

**Output:**

**c) Write a Program to read a Character value from STDIN into variable and print that value.**

**Program:**

```
#include <stdio.h>

int main( )
{
    char x;

    printf("\n\n Enter the value of x: ");
    scanf("%c", &x);

    printf("\n\n Value of x is: %c", x);

    return 0;
}
```

**Output:**

**d) Write a Program to read a Float value from STDIN into variable and print that value.**

**Program:**

```
#include <stdio.h>

int main( )
{
    float x;

    printf("\n\n Enter the value of x: ");
    scanf("%f", &x);

    printf("\n\n Value of x is: %f", x);

    return 0;
}
```

**Output:**

**e) Write a Program that depicts the concept of Arithmetic Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, add, sub, mul, div, mod;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    add = x + y;

    sub = x - y;

    mul = x * y;

    div = x / y;

    mod = x % y;

    printf("\n\n Addition of x and y is: %d", add);

    printf("\n\n Subtraction of x and y is: %d", sub);

    printf("\n\n Multiplication of x and y is: %d", mul);
```

```
printf("\n\n Division of x and y is: %d", div);

printf("\n\n Modulo Division of x and y is: %d", mod);

return 0;

}
```

**Output:**

**f) Write a Program that depicts the concept of Relational Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, gt, lt, gte, lte, eq, ne;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    gt = x > y;

    lt = x < y;

    gte = x >= y;

    lte = x <= y;

    eq = x == y;

    ne = x != y;

    printf("\n\n Value of Greater Than is: %d", gt);

    printf("\n\n Value of Less Than is: %d", lt);
```

```
printf("\n\n Value of Greater Than or Equal To is: %d", gte);

printf("\n\n Value of Less Than or Equal To is: %d", lte);

printf("\n\n Value of Equal To is: %d", eq);

printf("\n\n Value of Not Equal To is: %d", ne);

return 0;
}
```

**Output:**



**g) Write a Program that depicts the concept of Logical Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, z, and, or, not;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    printf("\n\n Enter the value of z: ");
    scanf("%d", &z);

    and = (x>y) && (x>z);
    or = (y>x) || (y>z);
    not = !(x>y);

    printf("\n\n Value of and is: %d", and);
    printf("\n\n Value of or is: %d", or);
    printf("\n\n Value of not is: %d", not);
    return 0;
}
```

**Output:**

**h) Write a Program that depicts the concept of Assignment Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, a;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    a = x;

    printf("\n\n Value of a is: %d", a);

    a+=10;

    printf("\n\n Value of a after Addition Assignment is: %d", a);

    a-=10;

    printf("\n\n Value of a after Subtraction Assignment is: %d", a);

    a*=10;

    printf("\n\n Value of a after Multiplication Assignment is: %d", a);

    a/=10;
```

```
printf("\n\n Value of a after Division Assignment is: %d", a);  
  
a%=10;  
  
printf("\n\n Value of a after Modulo Division Assignment is: %d", a);  
  
return 0;  
}
```

**Output:**

**i) Write a Program that depicts the concept of Bitwise Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, band, bor, bxor, bnot, blsh, brsh;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    band = x & y;
    printf("\n\n Bitwise AND of x and y is: %d", band);

    bor = x | y;
    printf("\n\n Bitwise OR of x and y is: %d", bor);

    bxor = x ^ y;
    printf("\n\n Bitwise XOR of x and y is: %d", bxor);

    bnot = ~x;
    printf("\n\n Bitwise NOT of x is: %d", bnot);

    blsh = x<<3;
    printf("\n\n Left Shift of x with 3 is: %d", blsh);
```

```
brsh = y>>3;
printf("\n\n Right Shift of y with 3 is: %d", brsh);

return 0;
}
```

**Output:**

**j) Write a Program that depicts the concept of Conditional and Unary Operators.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, cond, preinc, postinc, predec, postdec;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);
    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    cond = (x>y)?x:y;
    printf("\n\n Value of Condition Operator is: %d", cond);
    preinc = ++x;
    printf("\n\n Pre-Increment of x is: %d", preinc);
    postinc = y++;
    printf("\n\n Post-Increment of y is: %d", postinc);
    predec = --y;
    printf("\n\n Pre-Decrement of y is: %d", predec);
    postdec = x--;
    printf("\n\n Post-Decrement of x is: %d", postdec);
    return 0;
}
```

**Output:**

## **Module 4: Exercise Programs on Control Structures**

**a) Write a Program to check whether a given Number is EVEN or ODD.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int x;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    if(x%2 == 0)
    {
        printf("\n\n %d is EVEN NUMBER", x);
    }
    else
    {
        printf("\n\n %d is ODD NUMBER", x);
    }

    return 0;
}
```

**Output:**



**b) Write a Program to check biggest among three numbers.****Program:**

```
#include <stdio.h>

int main( )
{
    int x, y, z;

    printf("\n\n Enter the value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the value of y: ");
    scanf("%d", &y);

    printf("\n\n Enter the value of z: ");
    scanf("%d", &z);

    if(x>y && x>z)
    {
        printf("\n\n x = %d is BIGGEST NUMBER", x);
    }
    else if(y>x && y>z)
    {
        printf("\n\n y = %d is BIGGEST NUMBER", y);
    }
    else if(z>x && z>y)
    {
        printf("\n\n z = %d is BIGGEST NUMBER", z);
    }
}
```

```
    else
    {
        printf("\n\n Some of them are EQUAL");
    }

    return 0;
}
```

**Output:**

**c) Write a Program to check whether the entered year is Leap Year or Not.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int year;

    printf("\n\n Enter a year: ");
    scanf("%d", &year);

    if( (year % 4==0) && ( (year % 400 == 0) || (year % 100 != 0) ) )
    {
        printf("%d is a leap year", year);
    }
    else {
        printf("%d is not a leap year", year);
    }

    return 0;
}
```

**Output:**

**d) Write a Program to check the grade of the students based on marks.**

**Constraints:**

- If marks <50, then Grade is F
- if marks >=50 and <60, then Grade is D
- if marks >=60 and <70, then Grade is C
- if marks >=70 and <80, then Grade is B
- if marks >=80 and <90, then Grade is A
- if marks >=90, then Grade is A+

**Program:**

```
#include<stdio.h>

int main( )
{
    int marks;

    printf("\n\n Enter your marks: ");
    scanf("%d",&marks);

    if(marks<0 || marks>100)
    {
        printf("Wrong Entry");
    }
    else if(marks<50)
    {
        printf("Grade F");
    }
    else if(marks>=50 && marks<60)
    {
```

```
    printf("Grade D");  
}  
else if(marks>=60 && marks<70)  
{  
    printf("Grade C");  
}  
else if(marks>=70 && marks<80)  
{  
    printf("Grade B");  
}  
else if(marks>=80 && marks<90)  
{  
    printf("Grade A");  
}  
else  
{  
    printf("Grade A+");  
}  
  
return 0;  
}
```

**Output:**

**e) Write a Program to check the grade of the students based on average marks of six subjects.**

Mark Range	Grade
91-100	A1
81-90	A2
71-80	B1
61-70	B2
51-60	C1
41-50	C2
33-40	D
21-32	E1
0-20	E2

**Program:**

```
#include<stdio.h>

int main()
{
    int m1, m2, m3, m4, m5, m6, sum=0;
    float avg;

    printf("\n\n Enter Marks obtained in Subject 1: ");
    scanf("%d", &m1);

    printf("\n\n Enter Marks obtained in Subject 2: ");
    scanf("%d", &m2);

    printf("\n\n Enter Marks obtained in Subject 3: ");
    scanf("%d", &m3);

    printf("\n\n Enter Marks obtained in Subject 4: ");
    scanf("%d", &m4);
```

```
printf("\n\n Enter Marks obtained in Subject 5: ");  
scanf("%d", &m5);
```

```
printf("\n\n Enter Marks obtained in Subject 6: ");  
scanf("%d", &m6);
```

```
avg = sum/5;
```

```
printf("\n\n Grade = ");
```

```
if(avg>=91 && avg<=100)  
    printf("A1");
```

```
else if(avg>=81 && avg<91)  
    printf("A2");
```

```
else if(avg>=71 && avg<81)  
    printf("B1");
```

```
else if(avg>=61 && avg<71)  
    printf("B2");
```

```
else if(avg>=51 && avg<61)  
    printf("C1");
```

```
else if(avg>=41 && avg<51)  
    printf("C2");
```

```
    else if(avg>=33 && avg<41)
        printf("D");

    else if(avg>=21 && avg<33)
        printf("E1");

    else if(avg>=0 && avg<21)
        printf("E2");

    else
        printf("Invalid!");

    return 0;
}
```

**Output:**



**f) Write a Program to Calculate the Electricity Bill based on the following conditions:**

- For first 50 units Rs. 0.50/unit
- For next 100 units Rs. 0.75/unit
- For next 100 units Rs. 1.20/unit
- For unit above 250 Rs. 1.50/unit
- An additional surcharge of 20% is added to the bill

**Program:**

```
#include <stdio.h>

int main( )
{
    int unit;
    float amt, total_amt, sur_charge;

    printf("\n\n Enter total units consumed: ");
    scanf("%d", &unit);

    if(unit <= 50)
    {
        amt = unit * 0.50;
    }
    else if(unit <= 150)
    {
        amt = 25 + ((unit-50) * 0.75);
    }
    else if(unit <= 250)
    {
        amt = 100 + ((unit-150) * 1.20);
```

```
    }  
    else  
    {  
        amt = 220 + ((unit-250) * 1.50);  
    }  
  
    sur_charge = amt * 0.20;  
  
    total_amt = amt + sur_charge;  
  
    printf("\n\n Electricity Bill = Rs. %.2f", total_amt);  
  
    return 0;  
}
```

**Output:**

## **Module 5: Exercise Programs on Loops & nesting of Loops**

**a) Write a Program to print 1 to n Numbers using while loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i=1;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    while(i<=n)
    {
        printf("%d \n", i);
        i = i + 1;
    }

    return 0;
}
```

**Output:**

**b) Write a Program to print n to 1 Numbers using do-while loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    do
    {
        printf("%d \n", n);
        n = n - 1;
    }while(n>0);

    return 0;
}
```

**Output:**

**c) Write a Program to print EVEN Numbers between 1 to n using for loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    for(i=0; i<=n; i++)
    {
        if(i%2 == 0)
        {
            printf("%d \n", i);
        }
    }

    return 0;
}
```

**Output:**

**d) Write a Program to find the Number of digits in a Number.****Program:**

```
#include <stdio.h>

int main( )
{
    int n, count=0;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    while(n>0)
    {
        count = count + 1;
        n = n/10;
    }

    printf("\n\n No. of Digits in Given Number: %d", count);

    return 0;
}
```

**Output:**

**e) Write a Program to find the Sum of 1 to n numbers using for loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i, sum=0;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

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

    printf("\n\n Sum of 1 to n is: %d", sum);

    return 0;
}
```

**Output:**



**f) Write a Program to find the factorial of a number using while loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i=1, fact = 1;
    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);
    if(n<0)
    {
        printf("\n\n Invalid Input");
    }
    else if(n==0 || n==1)
    {
        printf("\n\n Factorial is: 1");
    }
    else
    {
        while(i<=n)
        {
            fact = fact * i;
            i = i + 1;
        }
        printf("\n\n Factorial of %d is: %d", n, fact);
    }
    return 0;
}
```

**Output:**

**g) Write a Program to find the factorial of a number using for loop.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i=1, fact = 1;
    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);
    if(n<0)
    {
        printf("\n\n Invalid Input");
    }
    else if(n==0 || n==1)
    {
        printf("\n\n Factorial is: 1");
    }
    else
    {
        for(i=1; i<=n; i++)
        {
            fact = fact * i;
        }
        printf("\n\n Factorial of %d is: %d", n, fact);
    }
    return 0;
}
```

**Output:**

**h) Write a Program to find the sum of digits of a number.****Program:**

```
#include <stdio.h>

int main( )
{
    int n, i=0, digit, sum=0;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    while(n>0)
    {
        digit = n % 10;
        sum = sum + digit;
        n = n/10;
    }

    printf("\n\n Sum of Digits of a Number is: %d", sum);

    return 0;
}
```

**Output:**

**i) Write a Program to find the count of EVEN Numbers and ODD Numbers in a Number.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, i=0, even = 0, odd = 0, digit;
    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    while(n>0)
    {
        digit = n%10;

        if(digit%2 == 0)
        {
            even = even + 1;
        }
        else
        {
            odd = odd + 1;
        }
        n = n/10;
    }
    printf("\n\n EVEN Digits in Number is: %d", even);
    printf("\n\n ODD Digits in Number is: %d", odd);
    return 0;
}
```

**Output:**

**j) Write a Program to check whether a Number is Armstrong Number or Not.**

**Program:**

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

int main( )
{
    int n, count = 0, i, temp1, temp2, rem, sum = 0;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    temp1 = n;

    temp2 = n;

    while(temp1>0)
    {
        count = count + 1;
        temp1 = temp1 / 10;
    }

    while(temp2>0)
    {
        rem = temp2 % 10;
        sum = sum + pow(rem, count);
        temp2 = temp2 / 10;
    }

    printf("\n\n Original Value of n is: %d", n);
```

```
printf("\n\n Sum of Powers of Digits: %d", sum);

if(sum == n)
{
    printf("\n\n %d is Armstrong Number", n);
}
else
{
    printf("\n\n %d is Not Armstrong Number", n);
}

return 0;
}
```

**Output:**

**k) Write a Program to check whether a Number is Prime Number or Not.**

**Program:**

```
#include <stdio.h>
int main( )
{
    int n, count = 0, i;
    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    for(i=1; i<=n; i++)
    {
        if(n%i == 0)
        {
            count = count + 1;
        }
    }

    if(count>2)
    {
        printf("\n\n %d is NOT A PRIME NUMBER", n);
    }
    else
    {
        printf("\n\n %d is A PRIME NUMBER", n);
    }
    return 0;
}
```

**Output:**

**I) Write a Program to check whether a Number is Strong Number or Not.****Program:**

```
#include <stdio.h>

int main( )
{
    int n, rem, fact=1, i, sum = 0, temp;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    temp = n;

    while(n>0)
    {
        rem = n % 10;

        for(i=1; i<=rem; i++)
        {
            fact = fact * i;
        }

        sum = sum + fact;

        fact = 1;

        n = n / 10;
    }

    if(temp == sum)
    {
        printf("\n\n %d is Strong Number", temp);
    }
}
```



```
    else
    {
        printf("\n\n %d is Not a Strong Number", temp);
    }

    return 0;
}
```

**Output:**

**m) Write a Program to check whether a Number is Perfect Number or Not.**

**Program:**

```
#include <stdio.h>
int main( )
{
    int n, i, sum = 0, temp;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    temp = n;

    for(i=1; i<n; i++)
    {
        if(n%i == 0)
        {
            sum = sum + i;
        }
    }

    if(sum == temp)
    {
        printf("\n\n %d is a Perfect Number", temp);
    }
    else
    {
        printf("\n\n %d is Not a Perfect Number", temp);
    }

    return 0;
}
```

**Output:**

**n) Write a Program to print the Fibonacci Series up to n Number.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, a=0, b=1, c, i;

    printf("\n\n Enter the Value of n: ");
    scanf("%d", &n);

    printf("%d \t %d \t", a, b);

    for(i=2; i<=n; i++)
    {
        c = a + b;
        printf("%d \t", c);
        a = b;
        b = c;
    }

    return 0;
}
```

**Output:**

**o) Write a Program to check whether the Number is in the series of Fibonacci or Not.**

**Program:**

```
#include <stdio.h>

int main( )

{

    int a, b, c, next, num;

    printf("Enter any number: ");
    scanf("%d", &num);

    if((num==0)|| (num==1))
    {
        printf("\n\n %d is a Number in Fibonacci Series",num);
    }

    else
    {
        a = 0;

        b = 1;

        c = a + b;

        while (c < num)
        {
            a = b;

            b = c;

            c = a + b;
        }

        if(c == num)
        {
            printf("\n\n %d is a Number in Fibonacci Series",num);
        }
    }
}
```

```
    else
    {
        printf("\n\n %d is Not a Number in Fibonacci Series",num);
    }
}

return 0;

}
```

**Output:**

**p) Write a Program to print the following Pattern:**

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

**Program:**

```
#include <stdio.h>  
  
int main( )  
{  
    int i, j, rows;  
  
    printf("Enter the number of rows: ");  
    scanf("%d", &rows);  
  
    for (i = 1; i <= rows; i++)  
    {  
        for (j = 1; j <= i; j++)  
        {  
            printf("* ");  
        }  
        printf("\n");  
    }  
    return 0;  
}
```

**Output:**

**q) Write a Program to print the following Pattern:**

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

**Program:**

```
#include <stdio.h>

int main( )
{
    int i, j, rows;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++)
    {
        for (j = 1; j <= i; j++)
        {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

**Output:**

**r) Write a Program to print the following Pattern:**

```
A
B B
C C C
D D D D
E E E E E
```

**Program:**

```
#include <stdio.h>

int main( )
{
    int i, j;
    char input, alphabet = 'A';

    printf("\n\n Enter an uppercase character you want to print in last row: ");
    scanf("%c", &input);

    for (i = 1; i <= (input - 'A' + 1); i++)
    {
        for (j = 1; j <= i; j++)
        {
            printf("%c ", alphabet);
        }

        ++alphabet;
        printf("\n\n");
    }

    return 0;
}
```

**Output:**



**s) Write a Program to print the following Pattern:**

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

**Program:**

```
#include <stdio.h>
int main()
{
    int i, j, rows;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = rows; i >= 1; i--)
    {
        for (j = 1; j <= i; j++)
        {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

**Output:**

**t) Write a Program to print the following Pattern:**

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

**Program:**

```
#include <stdio.h>

int main()
{
    int i, j, rows;
    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = rows; i >= 1; i--)
    {
        for (j = 1; j <= i; j++)
        {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

**Output:**

**u) Write a Program to print the following Pattern:**

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

**Program:**

```
#include <stdio.h>
int main()
{
    int i, space, rows, k = 0;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; ++i, k = 0)
    {
        for (space = 1; space <= rows - i; ++space)
        {
            printf(" ");
        }
        while (k != 2 * i - 1)
        {
            printf("* ");
            ++k;
        }
        printf("\n\n");
    }
    return 0;
}
```

**Output:**

v) Write a Program to print the following Pattern:

```
      1
    1  1
  1  2  1
1  3  3  1
1  4  6  4  1
1  5 10 10 5  1
```

**Program:**

```
#include <stdio.h>
int main()
{
    int rows, coef = 1, space, i, j;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 0; i < rows; i++)
    {
        for (space = 1; space <= rows - i; space++)
            printf(" ");
        for (j = 0; j <= i; j++)
        {
            if (j == 0 || i == 0)
                coef = 1;
            else
                coef = coef * (i - j + 1) / j;
            printf("%4d", coef);
        }
        printf("\n");
    }
    return 0;
}
```

**Output:**

**w) Write a Program to print the following Pattern:**

```
1
2 3
4 5 6
7 8 9 10
```

**Program:**

```
#include <stdio.h>
int main()
{
    int rows, i, j, number = 1;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (i = 1; i <= rows; i++)
    {
        for (j = 1; j <= i; ++j)
        {
            printf("%d ", number);
            ++number;
        }
        printf("\n\n");
    }
    return 0;
}
```

**Output:**

## **Module 6: Exercise Programs on Arrays & Strings**

**a) Write a Program that depicts the concept of initializing and accessing 1D Array.**

**Program:**

```
#include<stdio.h>

int main()
{

    int n, i, a[100];

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d Elements: ");
    for(i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("\n\n Elements of the Array are: \n\n");
    for(i=0; i<n; i++)
    {
        printf("%d ", a[i]);
    }
    return 0;
}
```

**Output:**

**b) Write a Program to sort the elements of an Array in Ascending Order (Bubble Sort).**

**Program:**

```
#include <stdio.h>

int main()
{
    int array[100], n, c, d, swap;

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for (c = 0; c < n; c++)
    {
        scanf("%d", &array[c]);
    }

    for (c = 0 ; c < n - 1; c++)
    {
        for (d = 0 ; d < n - c - 1; d++)
        {
            if (array[d] > array[d+1])
            {
                swap    =  array[d];

                array[d] =  array[d+1];

                array[d+1] =  swap;
            }
        }
    }

    printf("\n\n Sorted Elements in Ascending order:\n\n");
```



```
    for (c = 0; c < n; c++)  
    {  
        printf("%d ", array[c]);  
    }  
  
    return 0;  
  
}
```

**Output:**

**c) Write a Program to sort the elements of an Array in Descending Order (Bubble Sort).**

**Program:**

```
#include <stdio.h>

int main()
{
    int array[100], n, c, d, swap;

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for (c = 0; c < n; c++)
    {
        scanf("%d", &array[c]);
    }

    for (c = 0 ; c < n - 1; c++)
    {
        for (d = 0 ; d < n - c - 1; d++)
        {
            if (array[d] < array[d+1])
            {
                swap    =  array[d];

                array[d] =  array[d+1];

                array[d+1] =  swap;
            }
        }
    }

    printf("\n\n Sorted Elements in Descending order:\n\n");
```

```
for (c = 0; c < n; c++)  
{  
    printf("%d ", array[c]);  
}  
  
return 0;  
}
```

**Output:**

**d) Write a Program to search whether an element is present in an Array or Not (Linear Search).****Program:**

```
#include <stdio.h>

int main()
{
    int array[100], n, i, search, found=0;

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for (i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("\n\n Enter the Element to Search in the Array: ");
    scanf("%d", &search);

    for (i = 0; i < n; i++)
    {
        if(array[i] == search)
        {
            found = found + 1;

            break;
        }
    }

    if(found == 0)
    {
        printf("\n\n %d is NOT PRESENT in Array", search);
    }
}
```

```
    else
    {
        printf("\n\n %d is PRESENT in Array", search);
    }

    return 0;

}
```

**Output:**

**e) Write a Program to search whether an element is present in an Array or Not (Binary Search).**

**Program:**

```
#include <stdio.h>

int main( )
{

    int n, array[100], i, j, temp, search, low, high, mid;

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for(i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    for (i = 0 ; i < n - 1; i++)
    {

        for (j = 0 ; j < n - i - 1; j++)
        {

            if (array[j] > array[j+1])
            {
                temp    = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }

        }

    }

    printf("\n\n Enter the Element to Search in the Array: ");

    scanf("%d", &search);
```

```
low = 0;

high = n - 1;

mid = (low+high)/2;

while (low <= high)
{
    if(array[mid] < search)
    {
        low = mid + 1;
    }
    else if (array[mid] == search)
    {
        printf("\n\n %d is present in the array", search);
        break;
    }
    else
    {
        high = mid - 1;
    }

    mid = (low + high)/2;
}

if(low > high)
{
    printf("\n\n %d is not present in the array", search);
}

return 0;
}
```

**Output:**

**f) Write a Program to find the biggest and smallest element in an array.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, array[100], i, j, temp;

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for(i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    for (i = 0 ; i < n - 1; i++)
    {
        for (j = 0 ; j < n - i - 1; j++)
        {
            if (array[j] > array[j+1])
            {
                temp    = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }

    printf("\n\n Smallest Element of the Array: %d", array[0]);
    printf("\n\n Biggest Element of the Array: %d", array[n-1]);

    return 0;
}
```

**Output:**



**g) Write a Program to find the SUM of ODD NUMBERS and EVEN NUMBERS in an array.**

**Program:**

```
#include <stdio.h>

int main( )
{
    int n, array[100], i, odd_sum = 0, even_sum = 0;
    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);
    printf("\n\n Enter %d integers: ", n);

    for(i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }

    for(i=0; i<n; i++)
    {
        if(array[i] % 2 == 0)
        {
            even_sum = even_sum + array[i];
        }
        else
        {
            odd_sum = odd_sum + array[i];
        }
    }

    printf("\n\n Sum of EVEN Numbers of Array: %d", even_sum);
    printf("\n\n Sum of ODD Numbers of Array: %d", odd_sum);
    return 0;
}
```

**Output:**

**h) Write a Program to reverse the elements of an array.****Program:**

```
#include <stdio.h>

int main( )
{
    int n, a[100], i, j, b[100];

    printf("\n\n How many Elements you want to Store in Array? ");
    scanf("%d", &n);

    printf("\n\n Enter %d integers: ", n);

    for(i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("\n\n Original Array is: \n\n");

    for(i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }

    j=0;

    for(i=n-1; i>=0; i--)
    {
        b[j] = a[i];
        j++;
    }

    printf("\n\n Reverse Array is: \n\n");

    for(i = 0; i < n; i++)
    {
        printf("%d ", b[i]);
    }

    return 0;
}
```

**Output:**

**i) Write a Program that depicts the concept of initializing and accessing 2D Array.**

**Program:**

```
#include<stdio.h>

int main()
{
    int n, i, j, a[10][10], rows, cols;

    printf("\n\n How many Rows you want to Store in Array? ");
    scanf("%d", &rows);

    printf("\n\n How many Cols you want to Store in Array? ");
    scanf("%d", &cols);

    printf("\n\n Enter %d Elements: ", rows * cols);

    for(i=0; i<rows; i++)
    {
        for(j=0; j<cols; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }

    printf("\n\n Elements of the Array are: \n\n");
    for(i=0; i<rows; i++)
    {
        for(j=0; j<cols; j++)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n\n");
    }

    return 0;
}
```

**Output:**

**j) Write a Program to perform the Addition of Two Matrices using 2D Arrays.****Program:**

```
#include <stdio.h>

int main( )
{
    int a[100][100], b[100][100], res[100][100], ra, ca, rb, cb, i, j;

    printf("\n\n How many Rows you want to Store in Array A? ");
    scanf("%d", &ra);

    printf("\n\n How many Cols you want to Store in Array A? ");
    scanf("%d", &ca);

    printf("\n\n How many Rows you want to Store in Array B? ");
    scanf("%d", &rb);

    printf("\n\n How many Cols you want to Store in Array B? ");
    scanf("%d", &cb);

    if(ra!=rb || ca!=cb)
    {
        printf("\n\n Matrix Addition Not Possible");
    }
    else
    {
        printf("\n\n Enter %d integers into Array A: ", ra * ca);

        for(i = 0; i < ra; i++)
        {
            for(j = 0; j < ca; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }

        printf("\n\n Enter %d integers into Array B: ", rb * cb);
        for(i = 0; i < rb; i++)
        {
            for(j = 0; j < cb; j++)
            {
                scanf("%d", &b[i][j]);
            }
        }
    }
}
```

```
printf("\n\n Matrix A is: \n\n");

for(i = 0; i < ra; i++)
{
    for(j = 0; j < ca; j++)
    {
        printf("%d ", a[i][j]);
    }
    printf("\n\n");
}

printf("\n\n Matrix B is: \n\n");

for(i = 0; i < rb; i++)
{
    for(j = 0; j < cb; j++)
    {
        printf("%d ", b[i][j]);
    }
    printf("\n\n");
}

//Addition of Two Matrices Logic

for(i=0; i<ra; i++)
{
    for(j=0; j<ca; j++)
    {
        res[i][j] = a[i][j] + b[i][j];
    }
}

printf("\n\n Addition of Matrix A and B is: \n\n");

for(i = 0; i < ra; i++)
{
    for(j = 0; j < ca; j++)
    {
        printf("%d ", res[i][j]);
    }
    printf("\n\n");
}

}

return 0;
}
```

**Output:**

**k) Write a Program to perform the Subtraction of Two Matrices using 2D Arrays.****Program:**

```
#include <stdio.h>

int main( )
{
    int a[100][100], b[100][100], res[100][100], ra, ca, rb, cb, i, j;

    printf("\n\n How many Rows you want to Store in Array A? ");
    scanf("%d", &ra);

    printf("\n\n How many Cols you want to Store in Array A? ");
    scanf("%d", &ca);

    printf("\n\n How many Rows you want to Store in Array B? ");
    scanf("%d", &rb);

    printf("\n\n How many Cols you want to Store in Array B? ");
    scanf("%d", &cb);

    if(ra!=rb || ca!=cb)
    {
        printf("\n\n Matrix Subtraction Not Possible");
    }
    else
    {
        printf("\n\n Enter %d integers into Array A: ", ra * ca);

        for(i = 0; i < ra; i++)
        {
            for(j = 0; j < ca; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }

        printf("\n\n Enter %d integers into Array B: ", rb * cb);
        for(i = 0; i < rb; i++)
        {
            for(j = 0; j < cb; j++)
            {
                scanf("%d", &b[i][j]);
            }
        }
    }
}
```

```
printf("\n\n Matrix A is: \n\n");

for(i = 0; i < ra; i++)
{
    for(j = 0; j < ca; j++)
    {
        printf("%d ", a[i][j]);
    }
    printf("\n\n");
}

printf("\n\n Matrix B is: \n\n");

for(i = 0; i < rb; i++)
{
    for(j = 0; j < cb; j++)
    {
        printf("%d ", b[i][j]);
    }
    printf("\n\n");
}

//Subtraction of Two Matrices Logic

for(i=0; i<ra; i++)
{
    for(j=0; j<ca; j++)
    {
        res[i][j] = a[i][j] - b[i][j];
    }
}

printf("\n\n Subtraction of Matrix A and B is: \n\n");

for(i = 0; i < ra; i++)
{
    for(j = 0; j < ca; j++)
    {
        printf("%d ", res[i][j]);
    }
    printf("\n\n");
}

return 0;
}
```



**Output:**

**I) Write a Program to perform the Multiplication of Two Matrices using 2D Arrays.****Program:**

```
#include <stdio.h>

int main( )
{
    int a[100][100], b[100][100], res[100][100], ra, ca, rb, cb, i, j, k;

    printf("\n\n How many Rows you want to Store in Array A? ");
    scanf("%d", &ra);

    printf("\n\n How many Cols you want to Store in Array A? ");
    scanf("%d", &ca);

    printf("\n\n How many Rows you want to Store in Array B? ");
    scanf("%d", &rb);

    printf("\n\n How many Cols you want to Store in Array B? ");
    scanf("%d", &cb);

    if(ca != rb)
    {
        printf("\n\n Matrix Multiplication Not Possible");
    }
    else
    {
        printf("\n\n Enter %d integers into Array A: ", ra * ca);

        for(i = 0; i < ra; i++)
        {
            for(j = 0; j < ca; j++)
            {
                scanf("%d", &a[i][j]);
            }
        }

        printf("\n\n Enter %d integers into Array B: ", rb * cb);
        for(i = 0; i < rb; i++)
        {
            for(j = 0; j < cb; j++)
            {
                scanf("%d", &b[i][j]);
            }
        }
    }
}
```

```
printf("\n\n Matrix A is: \n\n");
for(i = 0; i < ra; i++)
{
    for(j = 0; j < ca; j++)
    {
        printf("%d ", a[i][j]);
    }
    printf("\n\n");
}

printf("\n\n Matrix B is: \n\n");
for(i = 0; i < rb; i++)
{
    for(j = 0; j < cb; j++)
    {
        printf("%d ", b[i][j]);
    }
    printf("\n\n");
}

//Matrix Multiplication Logic
for(i=0; i<ra; i++)
{
    for(j=0; j<cb; j++)
    {
        res[i][j] = 0;
        for(k=0; k<rb; k++)
        {
            res[i][j] = res[i][j] + (a[i][k] * b[k][j]);
        }
    }
}

printf("\n\n Multiplication of Matrix A and B is: \n\n");
for(i = 0; i < ra; i++)
{
    for(j = 0; j < cb; j++)
    {
        printf("%d ", res[i][j]);
    }
    printf("\n\n");
}
}

return 0;
}
```

**Output:**

**m) Write a Program to perform the Transpose of a Matrix using 2D Arrays.**

**Program:**

```
#include<stdio.h>
int main( )
{
    int x[10][20], row_size, col_size, i, j;

    printf("\n\n Enter Size of the Row: ");
    scanf("%d", &row_size);

    printf("\n\n Enter Size of the Column: ");
    scanf("%d", &col_size);

    printf("\n\n Enter %d Elements: ", row_size*col_size);
    for(i=0; i<row_size; i++)
    {
        for(j=0; j<col_size; j++)
        {
            scanf("%d", &x[i][j]);
        }
    }

    printf("\n\n Matrix Elements are: \n\n");
    for(i=0; i<row_size; i++)
    {
        for(j=0; j<col_size; j++)
        {
            printf("%d  ", x[i][j]);
        }
        printf("\n\n");
    }

    printf("\n\n Transpose Matrix Elements are: \n\n");
    for(i=0; i<col_size; i++)
    {
        for(j=0; j<row_size; j++)
        {
            printf("%d  ", x[j][i]);
        }
        printf("\n\n");
    }
    return 0;
}
```

**Output:**

**n) Write a Program to find the length of a String without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    char str[1000];
    int i, length = 0;

    printf("\n\n Enter a String: ");
    scanf("%s", str);

    for (i = 0; str[i] != '\0'; i++)
    {
        length = length + 1;
    }

    printf("\n\n Length of String is: %d", length);

    return 0;
}
```

**Output:**

**o) Write a Program to Reverse a String without using String Handling Function.****Program:**

```
#include <stdio.h>

int main()
{
    char str[1000], rev[1000];
    int i, j, length = 0;

    printf("\n\n Enter a String: ");
    scanf("%s", str);

    for (i = 0; str[i] != '\0'; i++)
    {
        length = length + 1;
    }

    //Reverse of a String Logic

    j=0;

    for(i=length-1; i>=0; i--)
    {
        rev[j] = str[i];
        j++;
    }

    printf("\n\n Original String is: %s", str);

    printf("\n\n Reverse String is: %s", rev);

    return 0;
}
```

**Output:**



**p) Write a Program to Copy a String to another String without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    char str[1000], copy[1000];
    int i, length = 0;

    printf("\n\n Enter a String: ");
    scanf("%s", str);

    for (i = 0; str[i] != '\0'; i++)
    {
        length = length + 1;
    }

    //Copying a String Logic

    for(i=0; i<length; i++)
    {
        copy[i] = str[i];
    }

    printf("\n\n Original String is: %s", str);

    printf("\n\n Copied String is: %s", copy);

    return 0;
}
```

**Output:**

**q) Write a Program to Concatenate a String to another String without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    int i, len=0, j;

    char str1[50], str2[50];

    printf("\n\n Enter First String : " );
    scanf("%s", str1);

    printf("\n\n Enter Second String : " );
    scanf("%s", str2);

    printf("\n\n First String is: %s", str1);
    printf("\n\n Second String is: %s", str2);

    for(i=0; str1[i]!='\0'; i++)
    {
        len = len + 1;
    }

    for(i=0; str2[i]!='\0'; i++)
    {
        str1[len++]=str2[i];
    }

    printf("\n\n Concatenated String is: %s ", str1);

    return 0;
}
```

**Output:**

**r) Write a Program to Compare Two Strings are Same or not without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    int i, len_str1=0, len_str2=0, count=0;
    char str1[50], str2[50];

    printf("\n\n Enter First String : " );
    scanf("%s", str1);

    printf("\n\n Enter Second String : " );
    scanf("%s", str2);

    printf("\n\n First String is: %s", str1);

    printf("\n\n Second String is: %s", str2);

    for(i=0; str1[i]!='\0'; i++)
    {
        len_str1 = len_str1 + 1;
    }

    for(i=0; str2[i]!='\0'; i++)
    {
        len_str2 = len_str2 + 1;
    }

    if(len_str1 != len_str2)
    {
        printf("\n\n Both the Strings are Not Same");
    }
    else
    {
        for(i=0; i<len_str1; i++)
        {
            if(str1[i] != str2[i])
            {
                count = count + 1;
                break;
            }
        }
    }
}
```

```
        if(count==0)
        {
            printf("\n\n Both the Strings are SAME");
        }
        else
        {
            printf("\n\n Both the Strings are NOT SAME");
        }
    }

    return 0;
}
```

**Output:**

**s) Write a Program to Convert the given String to Uppercase without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    int i, len_str1=0;

    char str1[50];

    printf("\n\n Enter a String : " );
    scanf("%s", str1);

    printf("\n\n Original String is: %s", str1);

    for(i=0; str1[i]!='\0'; i++)
    {
        len_str1 = len_str1 + 1;
    }

    for(i=0; str1[i]!='\0'; i++)
    {
        if(str1[i]>='a' && str1[i]<='z')
        {
            str1[i] = str1[i] - 32;
        }
    }

    printf("\n\n Uppercase String is: %s", str1);

    return 0;
}
```

**Output:**

**t) Write a Program to Convert the given String to Lowercase without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    int i, len_str1=0;

    char str1[50];

    printf("\n\n Enter a String : " );
    scanf("%s", str1);

    printf("\n\n Original String is: %s", str1);

    for(i=0; str1[i]!='\0'; i++)
    {
        len_str1 = len_str1 + 1;
    }

    for(i=0; str1[i]!='\0'; i++)
    {
        if(str1[i]>='A' && str1[i]<='Z')
        {
            str1[i] = str1[i] + 32;
        }
    }

    printf("\n\n Lowercase String is: %s", str1);

    return 0;
}
```

**Output:**

**u) Write a Program to check whether a String is Palindrome or Not without using String Handling Function.**

**Program:**

```
#include <stdio.h>

int main()
{
    int i, j, len_str1=0, count=0;

    char str1[50], str2[50];

    printf("\n\n Enter a String : " );
    scanf("%s", str1);

    printf("\n\n Original String is: %s", str1);

    for(i=0; str1[i]!='\0'; i++)
    {
        len_str1 = len_str1 + 1;
    }

    j=0;

    for(i=len_str1-1; i>=0; i--)
    {
        str2[j] = str1[i];
        j++;
    }

    printf("\n\n Reverse String is: %s", str2);

    for(i=0; i<len_str1; i++)
    {
        if(str1[i] != str2[i])
        {
            count = count + 1;
            break;
        }
    }

    if(count == 0)
    {
        printf("\n\n String is a Palindrome");
    }
}
```

```
    else
    {
        printf("\n\n String is NOT a Palindrome");
    }

    return 0;
}
```

**Output:**



**v) Write a Program that depicts the concept of String Handling Functions.****Program:**

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

int main()
{
    int len=0, cmp;

    char str1[50], str2[50], str3[50];

    printf("\n\n Enter String 1: ");
    scanf("%s", str1);

    printf("\n\n Enter String 2: ");
    scanf("%s", str2);

    len = strlen(str1);
    printf("\n\n Length of String 1 is: %d", len);

    strcpy(str3, str1);
    printf("\n\n String 3 is: %s", str3);

    cmp = strcmp(str3, str2);
    printf("\n\n Comparison is: %d", cmp);

    strlwr(str2);
    printf("\n\n Lowercase String 2 is: %s", str2);

   strupr(str1);
    printf("\n\n Lowercase String 1 is: %s", str1);

    return 0;
}
```

**Output:**

## **Module 7: Exercise Programs on Pointers**

**a) Write a Program to initialize a variable and access it with the help of a Pointer Variable.**

**Program:**

```
#include<stdio.h>

int main()
{
    int x, *p;

    p = &x;

    printf("\n\n Enter the Value of x: ");
    scanf("%d", &x);

    printf("\n\n Value of x is: %d", x);

    printf("\n\n Value of x through Pointer Variable: %d", *p);

    return 0;
}
```

**Output:**

**b) Write a Program that depicts the concept of Pointer Arithmetic.****Program:**

```
#include <stdio.h>

int main()
{
    int x, y, *p, *q, diff;
    p = &x;
    q = &y;

    printf("\n\n Enter the Value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the Value of y: ");
    scanf("%d", &y);

    printf("\n\n Address of x: %u", &x);

    printf("\n\n Value of x: %d", x);

    printf("\n\n Address of x through p is: %u", p);

    p = p + 5;
    printf("\n\n Increment of Pointer p is: %u", p);

    q = q - 2;
    printf("\n\n Decrement of Pointer q is: %u", q);

    diff = p - q;
    printf("\n\n Difference between Two Addresses: %d", diff);

    return 0;
}
```

**Output:**

**c) Write a Program that depicts the concept of Pointer Expression.****Program:**

```
#include <stdio.h>

int main()
{
    int x, y, *p, *q;
    p = &x;
    q = &y;

    printf("\n\n Enter the Value of x: ");
    scanf("%d", &x);

    printf("\n\n Enter the Value of y: ");
    scanf("%d", &y);

    //Arithmetic Operators
    printf("\n\n Addition is: %d", *p + *q);
    printf("\n\n Subtraction is: %d", *p - *q);
    printf("\n\n Multiplication is: %d", *p * *q);
    printf("\n\n Division is: %d", *p / *q);
    printf("\n\n Modulo Division is: %d", *p % *q);

    //Relational Operators
    printf("\n\n Greater than is: %d", *p > *q);
    printf("\n\n Less than is: %d", *p < *q);
    printf("\n\n Greater than Equal to is: %d", *p >= *q);
    printf("\n\n Less than Equal to is: %d", *p <= *q);
    printf("\n\n Equal to is: %d", *p == *q);
    printf("\n\n Not Equal to is: %d", *p != *q);

    //Logical Operators
    printf("\n\n Logical AND is: %d", (*p > 10) && (*q < 10));
    printf("\n\n Logical OR is: %d", (*p > 10) || (*q < 10));

    //Assignment Operators
    printf("\n\n Addition Assignment: %d", *p+=3);
    printf("\n\n Subtraction Assignment: %d", *q-=3);
    printf("\n\n Multiplication Assignment: %d", *p*=3);
    printf("\n\n Division Assignment: %d", *q/=3);
    printf("\n\n Modulo Division Assignment: %d", *p%=3);

    return 0;
}
```

**Output:**

**d) Write a Program to initialize and access elements of 1D Array using Pointers.****Program:**

```
#include <stdio.h>

int main()
{
    int n, i, a[100];
    int *p;

    p = &a;

    printf("\n\n How many Elements you want to store? ");
    scanf("%d", &n);

    printf("\n\n Enter %d Elements into Array: ", n);
    for(i=0; i<n; i++)
    {
        scanf("%d", p+i);
    }

    printf("\n\n Array Elements are: ");
    for(i=0; i<n; i++)
    {
        printf("%d ", *(p+i));
    }

    return 0;
}
```

**Output:**

**e) Write a Program to initialize and access elements of 2D Array using Pointers.****Program:**

```
#include <stdio.h>

int main()
{
    int ra, ca, i, j, a[100][100];
    int *p;

    p = &a;

    printf("\n\n How many Rows you want in Matrix A? ");
    scanf("%d", &ra);

    printf("\n\n How many Cols you want in Matrix A? ");
    scanf("%d", &ca);

    printf("\n\n Enter %d Elements into Matrix A: ", ra * ca);

    for(i=0; i<ra; i++)
    {
        for(j=0; j<ca; j++)
        {
            scanf("%d", (p+i*ca)+j);
        }
    }

    printf("\n\n Array A Elements are: \n\n");
    for(i=0; i<ra; i++)
    {
        for(j=0; j<ca; j++)
        {
            printf("%d  ", *(p+i*ca)+j);
        }
        printf("\n\n");
    }

    return 0;
}
```



**Output:**

**f) Write a Program to perform Addition of TWO Matrices using Array of Pointers.****Program:**

```
#include <stdio.h>

int main()
{
    int ra, ca, rb, cb, i, j, a[100][100], b[100][100], res[100][100];

    int (*p)[6], (*q)[6];

    p = &a;
    q = &b;

    printf("\n\n How many Rows you want in Matrix A? ");
    scanf("%d", &ra);

    printf("\n\n How many Cols you want in Matrix A? ");
    scanf("%d", &ca);

    printf("\n\n How many Rows you want in Matrix B? ");
    scanf("%d", &rb);

    printf("\n\n How many Cols you want in Matrix B? ");
    scanf("%d", &cb);

    if(ra != rb || ca != cb)
    {
        printf("\n\n Matrix Addition Not Possible");
    }
    else
    {
        printf("\n\n Enter %d Elements into Matrix A: ", ra * ca);

        for(i=0; i<ra; i++)
        {
            for(j=0; j<ca; j++)
            {
                scanf("%d", *(p+i)+j);
            }
        }

        printf("\n\n Enter %d Elements into Matrix B: ", rb * cb);
```

```
for(i=0; i<rb; i++)
{
    for(j=0; j<cb; j++)
    {
        scanf("%d", *(q+i)+j);
    }
}

printf("\n\n Array A Elements are: \n\n");
for(i=0; i<ra; i++)
{
    for(j=0; j<ca; j++)
    {
        printf("%d  ", (*(p+i)+j));
    }
    printf("\n\n");
}

printf("\n\n Array B Elements are: \n\n");
for(i=0; i<rb; i++)
{
    for(j=0; j<cb; j++)
    {
        printf("%d  ", (*(q+i)+j));
    }
    printf("\n\n");
}

//Matrix Addition Logic
for(i=0; i<ra; i++)
{
    for(j=0; j<ca; j++)
    {
        (*(res+i)+j) = (*(p+i)+j) + (*(q+i)+j);
    }
    printf("\n\n");
}
```

```
printf("\n\n Addition of Two Matrices are: \n\n");

for(i=0; i<ra; i++)
{
    for(j=0; j<ca; j++)
    {
        printf("%d  ", (*(res+i)+j));
    }
    printf("\n\n");
}

return 0;
}
```

**Output:**

**g) Write a Program to print a String using Character Array with Pointer.****Program:**

```
#include<stdio.h>

int main ()
{
    int i, str_len=0;
    char *ptr;
    char str[100];

    printf ("\n\n Enter a String: ");
    scanf ("%s", str);

    ptr = &str;

    for(i=0; str[i]!='\0'; i++)
    {
        str_len = str_len + 1;
    }

    printf ("\n\n String using Pointer is: ");
    for (i = 0; i < str_len; i++)
    {
        printf ("%c", ptr[i]);
    }

    return 0;
}
```

**Output:**

**h) Write a Program that depicts the concept of Pointer to Pointer.**

**Program:**

```
#include<stdio.h>

int main ()
{
    int n;
    int *p, **q, ***r, ****s;

    printf ("\n\n Enter a Number: ");
    scanf ("%d", &n);

    p = &n;

    q = &p;

    r = &q;

    s = &r;

    printf ("\n\n Value of n through p: %d", *p);
    printf ("\n\n Value of n through q: %d", **q);
    printf ("\n\n Value of n through r: %d", ***r);
    printf ("\n\n Value of n through s: %d", ****s);

    return 0;
}
```

**Output:**

**i) Write a Program that depicts the concept of malloc( ) in Dynamic Memory Allocation.****Program:**

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

int main()
{
    int *ptr;

    int n, i;

    printf("\n\n How many Elements you want to store? ");
    scanf("%d", &n);

    // Dynamically allocate memory using malloc()
    ptr = (int *)malloc(n * sizeof(int));

    if (ptr == NULL)
    {
        printf("\n\n Memory not allocated");
        exit(0);
    }
    else
    {
        printf("\n\n Memory successfully allocated using malloc");

        // Get the elements of the array

        printf("\n\n Enter %d Elements: ", n);
        for (i = 0; i < n; i++)
        {
            scanf("%d", (ptr+i));
        }

        // Print the elements of the array

        printf("\n\n Elements of the array are: ");

        for (i = 0; i < n; ++i)
        {
            printf("%d ", *(ptr+i));
        }
    }
}
```

```
free(ptr);  
  
return 0;  
}
```

**Output:**



**j) Write a Program that depicts the concept of calloc( ) in Dynamic Memory Allocation.**

**Program:**

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

int main()
{
    int *ptr;

    int n, i;

    printf("\n\n How many Elements you want to store? ");
    scanf("%d", &n);

    // Dynamically allocate memory using malloc()
    ptr = (int *)calloc(n, sizeof(int));

    if (ptr == NULL)
    {
        printf("\n\n Memory not allocated");
        exit(0);
    }
    else
    {
        printf("\n\n Memory successfully allocated using calloc");

        // Get the elements of the array

        printf("\n\n Enter %d Elements: ", n);
        for (i = 0; i < n; i++)
        {
            scanf("%d", (ptr+i));
        }

        // Print the elements of the array

        printf("\n\n Elements of the array are: ");

        for (i = 0; i < n; ++i)
        {
            printf("%d ", *(ptr+i));
        }
    }
}
```

```
free(ptr);  
  
return 0;  
}
```

**Output:**

**k) Write a Program that depicts the concept of realloc( ) in Dynamic Memory Allocation.**

**Program:**

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

int main()
{
    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using calloc()
    ptr = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }

        // Get the new size for the array
```

```
n = 10;
printf("\n\nEnter the new size of the array: %d\n", n);

// Dynamically re-allocate memory using realloc()
ptr = realloc(ptr, n * sizeof(int));

// Memory has been successfully allocated
printf("Memory successfully re-allocated using realloc.\n");

// Get the new elements of the array
for (i = 5; i < n; ++i) {
    ptr[i] = i + 1;
}

// Print the elements of the array
printf("The elements of the array are: ");
for (i = 0; i < n; ++i) {
    printf("%d, ", ptr[i]);
}

free(ptr);
}

return 0;
}
```

**Output:**

## **Module 8: Exercise Programs on Functions**

**a) Write a Program to print 1 to n Numbers using User-Defined Functions (Without Arguments, Without Return Values).**

**Program:**

```
#include <stdio.h>

void displaynumb();

int main()
{
    displaynumb();

    return 0;
}

void displaynumb()
{
    int n, i;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    printf("\n\n Numbers are: \n\n");
    for(i=1; i<=n; i++)
    {
        printf("%d ", i);
    }

}
```

**Output:**

**b) Write a Program to print 1 to n Numbers using User-Defined Functions (With Arguments, Without Return Values).**

**Program:**

```
#include <stdio.h>

void displaynumb(int x);

int main()
{
    int n;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    displaynumb(n);

    return 0;
}

void displaynumb(int x)
{
    int i;

    printf("\n\n Numbers are: \n\n");
    for(i=1; i<=x; i++)
    {
        printf("%d ", i);
    }
}
```

**Output:**

**c) Write a Program to find factorial of a Number using User-Defined Functions (Without Arguments, With Return Values).****Program:**

```
#include <stdio.h>

int factorial();

int main()
{
    int res;

    res = factorial();

    printf("\n\n Factorial is: %d", res);

    return 0;
}

int factorial()
{
    int i, n, fact=1;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    for(i=1; i<=n; i++)
    {
        fact = fact * i;
    }

    return fact;
}
```

**Output:**



**d) Write a Program to find factorial of a Number using User-Defined Functions (With Arguments, With Return Values).****Program:**

```
#include <stdio.h>

int factorial(int x);

int main()
{
    int res, n;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    res = factorial(n);

    printf("\n\n Factorial is: %d", res);

    return 0;
}

int factorial(int x)
{
    int fact=1, i;

    for(i=1; i<=x; i++)
    {
        fact = fact * i;
    }

    return fact;
}
```

**Output:**

**e) Write a Program to SWAP TWO Numbers using User-Defined Functions (Call-by-Value Concept).****Program:**

```
#include <stdio.h>

void swap(int x, int y);

int main()
{
    int a, b;

    printf("\n\n Enter the value of a: ");
    scanf("%d", &a);

    printf("\n\n Enter the value of b: ");
    scanf("%d", &b);

    printf("\n\n Before Swapping, a=%d and b=%d", a, b);

    swap(a, b);

    return 0;
}

void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;

    printf("\n\n After Swapping, a=%d and b=%d", x, y);
}
```

**Output:**

**f) Write a Program to SWAP TWO Numbers using User-Defined Functions (Call-by-Reference Concept).****Program:**

```
#include <stdio.h>

void swap(int *x, int *y);

int main()
{
    int a, b;

    printf("\n\n Enter the value of a: ");
    scanf("%d", &a);

    printf("\n\n Enter the value of b: ");
    scanf("%d", &b);

    printf("\n\n Before Swapping, a=%d and b=%d", a, b);

    swap(&a, &b);

    printf("\n\n After Swapping, a=%d and b=%d", a, b);

    return 0;
}

void swap(int *x, int *y)
{
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;
}
```

**Output:**

**g) Write a Program to print SUM of 1 to n Natural Numbers using Recursion.**

**Program:**

```
#include <stdio.h>

int sum(int x);

int main()
{
    int n, res;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    res = sum(n);

    printf("\n\n Sum of 1 to %d is: %d", n, res);

    return 0;
}

int sum(int x)
{
    if(x==0)
    {
        return 0;
    }
    else
    {
        return x + sum(x-1);
    }
}
```

**Output:**

**h) Write a Program to print Factorial of a Number using Recursion.****Program:**

```
#include <stdio.h>

int fact(int x);

int main()
{
    int n, res;

    printf("\n\n Enter the value of n: ");
    scanf("%d", &n);

    res = fact(n);

    printf("\n\n Factorial of %d is: %d", n, res);

    return 0;
}

int fact(int x)
{
    if(x==0 || x==1)
    {
        return 1;
    }
    else
    {
        return x * fact(x-1);
    }
}
```

**Output:**

**i) Write a Program to print GCD of TWO Numbers using Recursion.****Program:**

```
#include <stdio.h>

int gcd(int x, int y);

int main()
{
    int a, b, res;

    printf("\n\n Enter the value of a: ");
    scanf("%d", &a);

    printf("\n\n Enter the value of b: ");
    scanf("%d", &b);

    res = gcd(a, b);

    printf("\n\n GCD of %d and %d is: %d", a, b, res);

    return 0;
}

int gcd(int x, int y)
{
    if(x==0)
    {
        return y;
    }

    else if(y==0)
    {
        return x;
    }

    else if(x>y)
    {
        return gcd(x%y, y);
    }
}
```

```
    else  
    {  
        return gcd(y, x%y);  
    }  
}
```

**Output:**

**j) Write a Program to calculate the sum of array elements by passing to a function.**

**Program:**

```
#include <stdio.h>
float calculateSum(float num[]);

int main()
{
    float result, num[] = {23.4, 55, 22.6, 3, 40.5, 18};

    // num array is passed to calculateSum()
    result = calculateSum(num);

    printf("Result = %.2f", result);

    return 0;
}

float calculateSum(float num[])
{
    float sum = 0.0;

    for (int i = 0; i < 6; ++i)
    {
        sum += num[i];
    }

    return sum;
}
```

**Output:**



## **Module 9: Exercise Programs on user defined data types**

**a) Write a Program that depicts the concept of structure declaration, initialization and accessing of structure members.**

**Program: Initializing with Dot Operator:**

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

struct employee
{
    int emp_id;
    char emp_name[10];
    float salary;
};
int main( )
{
    struct employee emp;

    emp.emp_id = 9848;
    strcpy(emp.emp_name, "Ramu");
    emp.salary = 9864.25;

    printf("\n\n ID is: %d", emp.emp_id);

    printf("\n\n Name is: %s", emp.emp_name);

    printf("\n\n Salary is: %f", emp.salary);

    return 0;
}
```

**Output:**

**Program: Value initialized structure variable:**

```
#include<stdio.h>

#include<string.h>

struct employee
{
    int emp_id;
    char emp_name[10];
    float salary;
};

int main()
{
    struct employee emp = {9848, "Ramu", 9864.25};

    printf("\n\n ID is: %d", emp.emp_id);

    printf("\n\n Name is: %s", emp.emp_name);

    printf("\n\n Salary is: %f", emp.salary);

    return 0;
}
```

**Output:**

**b) Write a Program that depicts the concept of storing 3 employee details without using arrays.**

**Program:**

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

struct employee
{
    int emp_id;
    char emp_name[10];
    float salary;
};

int main()
{
    struct employee emp1, emp2, emp3; //creating 3 reference variables for
    storing 3 employee details

    printf("\n\n Enter EMP_1 ID: ");
    scanf("%d", &emp1.emp_id);

    printf("\n\n Enter EMP_1 Name: ");
    scanf("%s", emp1.emp_name);

    printf("\n\n Enter EMP_1 Salary: ");
    scanf("%f", &emp1.salary);

    printf("\n\n Enter EMP_2 ID: ");
    scanf("%d", &emp2.emp_id);

    printf("\n\n Enter EMP_2 Name: ");
    scanf("%s", emp2.emp_name);

    printf("\n\n Enter EMP_2 Salary: ");
    scanf("%f", &emp2.salary);
```

```
printf("\n\n Enter EMP_3 ID: ");
scanf("%d", &emp3.emp_id);

printf("\n\n Enter EMP_3 Name: ");
scanf("%s", emp3.emp_name);

printf("\n\n Enter EMP_3 Salary: ");
scanf("%f", &emp3.salary);

printf("\n\n EMP_1 ID is: %d", emp1.emp_id);
printf("\n\n EMP_1 Name is: %s", emp1.emp_name);
printf("\n\n EMP_1 Salary is: %f", emp1.salary);

printf("\n\n EMP_2 ID is: %d", emp2.emp_id);
printf("\n\n EMP_2 Name is: %s", emp2.emp_name);
printf("\n\n EMP_2 Salary is: %f", emp2.salary);

printf("\n\n EMP_3 ID is: %d", emp3.emp_id);
printf("\n\n EMP_3 Name is: %s", emp3.emp_name);
printf("\n\n EMP_3 Salary is: %f", emp3.salary);

return 0;
}
```

**Output:**

**c) Write a Program to store Employee Data such as Employee ID, Employee Name, Employee Address into a Structure. Where Employee Address should be taken as a Structure which should be accessed from Employee Structure.**

**Program:**

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

struct emp_details
{
    int emp_id;
    char emp_name[10];
    struct address
    {
        int street_number;
        char city[10];
        int pin_code;
    }addr;
}emp;

int main()
{
    printf("\n\n Enter Employee ID: ");
    scanf("%d", &emp.emp_id);

    printf("\n\n Enter Name: ");
    scanf("%s", emp.emp_name);

    printf("\n\n Enter Emp Street: ");
    scanf("%d", &emp.addr.street_number);

    printf("\n\n Enter Emp City: ");
    scanf("%s", emp.addr.city);

    printf("\n\n Enter Emp Pin Code: ");
```

```
scanf("%d", &emp.addr.pin_code);

printf("\n\n*****Employee Details*****");

printf("\n\n Employee ID: %d", emp.emp_id);
printf("\n\n Employee Name: %s", emp.emp_name);
printf("\n\n Street Number: %d", emp.addr.street_number);
printf("\n\n City is: %s", emp.addr.city);
printf("\n\n Pin Code is: %d", emp.addr.pin_code);

return 0;
}
```

**Output:**

**d) Write a Program to create a Nested Structure with the Outer Structure Members as Student ID as integer, Student Name as a character, Student Address as an Inner Structure. Then the Inside Structure should have the following members i.e., Student Pincode, Student Street Number, Student District, and Student State.**

**Program:**

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

struct student
{
    int stud_id;
    char stud_name[10];
    struct studentaddress
    {
        int street_number;
        char dist[10], state[15];
        int pin_code;

    }addr;
}stud;

int main()
{

    printf("\n\n Enter Student ID: ");
    scanf("%d", &stud.stud_id);

    printf("\n\n Enter Name: ");
    scanf("%s", stud.stud_name);

    printf("\n\n Enter Street Number: ");
    scanf("%d", &stud.addr.street_number);
```



```
printf("\n\n Enter District: ");
scanf("%s", stud.addr.dist);

printf("\n\n Enter State: ");
scanf("%s", stud.addr.state);

printf("\n\n Enter Pin Code: ");
scanf("%d", &stud.addr.pin_code);

printf("\n\n*****Student Details*****");

printf("\n\n Student ID: %d", stud.stud_id);
printf("\n\n Employee Name: %s", stud.stud_name);
printf("\n\n Street Number: %d", stud.addr.street_number);
printf("\n\n City is: %s", stud.addr.dist);
printf("\n\n State is: %s", stud.addr.state);
printf("\n\n Pin Code is: %d", stud.addr.pin_code);

return 0;
}
```

**Output:**

e) Write a Program to create a Nested Structure with the Outer Structure Members as an Employee ID as integer, Employee Name as a character, Employee Total Salary as Float, Employee Salary as an Inner Structure. Then the Inside Structure should have the following members i.e., Employee Basic, Employee DA, Employee HRA.

Calculate the Total Salary as:

- a) Basic
- b) DA = 180% of Basic
- c) HRA = 15% of Basic

Display all the details of an Employee after Calculating the Total Salary.

**Program:**

```
#include<stdio.h>
#include<string.h>
struct emp_details
{
    int emp_id;
    char emp_name[10];
    float emp_total_sal;
    struct salary
    {
        int basic;
        float da, hra;
    }sal;
}emp;

int main()
{
    printf("\n\n Enter Employee ID: ");
    scanf("%d", &emp.emp_id);

    printf("\n\n Enter Name: ");
    scanf("%s", emp.emp_name);
```

```
printf("\n\n Enter Emp Basic Salary: ");
scanf("%d", &emp.sal.basic);

emp.sal.da = emp.sal.basic * 1.8;

emp.sal.hra = emp.sal.basic * 0.15;

emp.emp_total_sal = emp.sal.basic + emp.sal.da + emp.sal.hra;

printf("\n\n*****Employee Details*****");

printf("\n\n Employee ID: %d", emp.emp_id);
printf("\n\n Employee Name: %s", emp.emp_name);
printf("\n\n Employee Salary is: %.2f", emp.emp_total_sal);

return 0;
}
```

**Output:**

**f) Write a Program to store n student details such as Student Roll Number, Student Name, Student Percentage, etc. in Structures.**

**Program:**

```
#include<stdio.h>
struct stud
{
    int stud_id;
    char stud_name[15];
    float stud_percentage;
}s[20];
int main()
{
    int i, n;
    printf("\n\n Enter No. of Students: ");
    scanf("%d", &n);

    printf("\n\n Enter %d Student Details:\n\n", n);

    for(i=0; i<n; i++)
    {
        printf("\n\n Enter Student %d Roll Number: ", i+1);
        scanf("%d", &s[i].stud_id);

        printf("\n\n Enter Student %d Name: ", i+1);
        scanf("%s", s[i].stud_name);

        printf("\n\n Enter Student %d Percentage: ", i+1);
        scanf("%f", &s[i].stud_percentage);
    }

    printf("\n\n *****%d STUDENT DETAILS*****\n\n", n);
    for(i=0; i<n; i++)
    {
        printf("\n\n Student %d Roll Number is: %d", i+1, s[i].stud_id);
```

```
        printf("\n\n Student %d Name is: %s", i+1, s[i].stud_name);  
        printf("\n\n      Student      %d      Percentage      is:      %.2f",      i+1,  
s[i].stud_percentage);  
    }  
    return 0;  
}
```

**Output:**

**g) Write a Program to Pass entire structure as an argument to a function.****Program:**

```
#include <stdio.h>

struct student
{
    char name[50];
    int age;
};

void display (struct student); //user-defined function prototype

int main( )
{
    struct student s1;

    printf("\n\n Enter name: ");
    scanf("%s", s1.name);

    printf("\n\n Enter age: ");
    scanf("%d", &s1.age);

    display(s1); // calling function 2 struct s1 passed as argument to user-defined
function

    printf("\n\n");

    return 0;
}

void display (struct student s) // s1 in calling function is copied to s in called
function
{
```

```
printf("\n\n*****Student Information*****\n\n");  
printf("\n\n Name: %s", s.name);  
printf("\n\n Age: %d", s.age);  
}
```

**Output:**

**h) Write a Program to Pass structure member (value) as an argument to a function.**

**Program:**

```
#include <stdio.h>

struct student
{
    char name[50];
    int age;
};

void show (int, char studName[ ]); //user-defined function prototype with
formal arguments

int main( )
{
    struct student s1;

    printf("\n\n Enter name: ");
    scanf("%s", s1.name);

    printf("\n\n Enter age: ");
    scanf("%d", &s1.age);

    show(s1.age, s1.name); //passing structure members as arguments to called
function

    printf("\n\n");
    return 0;
}

void show (int studAge, char studName[ ])
{
```



```
printf("\n\n*****Student Information*****\n\n");  
printf("\n\n Name: %s", studName);  
printf("\n\n Age: %d", studAge);  
}
```

**Output:**

**i) Write a Program to Pass structure member (value) as an argument to a function.**

**Program:**

```
#include <stdio.h>

struct student
{
    char name[50];
    int age;
};

void show (int, char studName[ ]); //user-defined function prototype with
formal arguments

int main( )
{
    struct student s1;

    printf("\n\n Enter name: ");
    scanf("%s", s1.name);

    printf("\n\n Enter age: ");
    scanf("%d", &s1.age);

    show(s1.age, s1.name); //passing structure members as arguments to called
function

    printf("\n\n");
    return 0;
}

void show (int studAge, char studName[ ])
{
```

```
printf("\n\n*****Student Information*****\n\n");  
printf("\n\n Name: %s", studName);  
printf("\n\n Age: %d", studAge);  
}
```

**Output:**

**j) Write a Program to Pass structure member (reference) as an argument to a function.**

**Program:**

```
#include <stdio.h>

struct student
{
    char name[50];
    int age;
};

void disp(int *, char *); //user-defined function prototype with formal arguments

int main()
{
    struct student s1;

    printf("\n\n Enter name: ");
    scanf("%s", s1.name);

    printf("\n\n Enter age: ");
    scanf("%d", &s1.age);

    disp(&s1.age, &s1.name); //passing addresses of members as arguments to called function

    printf("\n\n");
    return 0;
}

void disp (int *studAge, char *studName)
{
```

```
printf("\n\n*****Student Information*****\n\n");  
printf("\n\n Name: %s", studName);  
printf("\n\n Age: %d", *studAge);  
}
```

**Output:**

**k) Write a Program to store employee details using arrow ( -> ) operator in Structures.**

**Program:**

```
#include<stdio.h>
struct employee
{
    int emp_id;
    char emp_name[15];
    float emp_salary;
};
int main( )
{
    struct employee *emp_ptr, emp;
    emp_ptr = &emp;

    printf("\n\n Enter Employee ID: ");
    scanf("%d", &emp_ptr->emp_id);

    printf("\n\n Enter Employee Name: ");
    scanf("%s", &emp_ptr->emp_name);

    printf("\n\n Enter Employee Salary: ");
    scanf("%f", &emp_ptr->emp_salary);

    printf("\n\n Employee ID: %d", emp_ptr->emp_id);
    printf("\n\n Employee Name: %s", emp_ptr->emp_name);
    printf("\n\n Employee Salary: %.2f", emp_ptr->emp_salary);
    return 0;
}
```

**Output:**

**I) Write a Program to pass structure to a function using Pointers.****Program:**

```
#include<stdio.h>

struct stud
{
    int stud_roll;
    char stud_name[15];
    float stud_per;
};

void showstudent (struct stud *);

int main()
{
    struct stud st;

    printf("\n\n Enter Student Roll Number: ");
    scanf("%d", &st.stud_roll);

    printf("\n\n Enter Student Name: ");
    scanf("%s", st.stud_name);

    printf("\n\n Enter Student Percentage: ");
    scanf("%f", &st.stud_per);

    showstudent(&st); //passing address of a structure as a parameter

    printf("\n\n");
    return 0;
}

void showstudent(struct stud *studpoint) // called function takes the address in
calling function and dereferenced to studpoint
{
```

```
printf("\n\n Student Roll Number: %d", studpoint->stud_roll);  
printf("\n\n Student Name: %s", studpoint->stud_name);  
printf("\n\n Student Percentage: %.2f", studpoint->stud_per);  
}
```

**Output:**



**m) Write a program to create a linked list with 3 nodes using Self-Referential Structures.**

**Program:**

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

struct node
{
    int data; //this is for storing data part of the node
    struct node *address; // this is for storing the address of the next node
};

int main( )
{
    struct node *node_1 = NULL;
    struct node *node_2 = NULL;
    struct node *node_3 = NULL;

    node_1 = (struct node *) malloc (sizeof(struct node));
    node_2 = (struct node *) malloc (sizeof(struct node));
    node_3 = (struct node *) malloc (sizeof(struct node));

    node_1->data = 10; //data of node_1
    node_2->data = 20; //data of node_2
    node_3->data = 30; //data of node_3

    node_1 -> address = node_2; //address of node_2 storing in address part of
node_1
    node_2 -> address = node_3; //address of node_3 storing in address part of
node_2
    node_3 -> address = NULL; //node_3 is the last node. node_3 doesn't have
address.

    printf("\n\n Data in node_1: %d", node_1->data);
```

```
printf("\n\n Data in node_2: %d", node_2->data);
```

```
printf("\n\n Data in node_3: %d", node_3->data);
```

```
return 0;
```

```
}
```

**Output:**

**n) Write a program to create a linked list with 3 nodes using Self-Referential Structures and access all the nodes using node\_1.**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data; //this is for storing data part of the node
    struct node *address; // this is for storing the address of the next node
};
int main( )
{
    struct node *node_1 = NULL;
    struct node *node_2 = NULL;
    struct node *node_3 = NULL;

    node_1 = (struct node *) malloc (sizeof(struct node));
    node_2 = (struct node *) malloc (sizeof(struct node));
    node_3 = (struct node *) malloc (sizeof(struct node));

    node_1->data = 10; //data of node_1
    node_2->data = 20; //data of node_2
    node_3->data = 30; //data of node_3

    node_1 -> address = node_2; //address of node_2 storing in address part of
node_1
    node_2 -> address = node_3; //address of node_3 storing in address part of
node_2
    node_3 -> address = NULL; //node_3 is the last node. node_3 doesn't have
address.

    printf("\n\n Data in node_1: %d", node_1->data);

    printf("\n\n Data in node_2: %d", node_1->address->data);
```

```
printf("\n\n Data in node_3: %d", node_1->address->address->data);  
  
printf("\n\n");  
return 0;  
}
```

**Output:**

**o) Write a Program that depicts the concept of typedef in C.**

**Program:**

```
#include <stdio.h>
int main( )
{
    typedef int ram;
    ram i, j;
    i=38;
    j=96;
    printf("\n\n Value of i is: %d", i);
    printf("\n\n Value of j is: %d", j);
    return 0;
}
```

**Output:**

## **Module 10: Exercise Programs on Files**

**a) Write a Program that reads the character by character from the file.**

**Program:**

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

int main()
{
    FILE *fp;
    char ch;

    fp = fopen("fpr.txt", "r");

    if(fp == NULL)
    {
        printf("\n\n FILE NOT FOUND\n\n");
        exit(0);
    }
    else
    {
        printf("\n\n FILE FOUND\n\n");
        while((ch=fgetc(fp))!=EOF)
        {
            fprintf(stdout, "%c", ch);
        }
    }
    fclose(fp);
    printf("\n\n");
    return 0;
}
```

**Output:**

**b) Write a Program that reads the entire line from a file and write to the stdout.**

**Program:**

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

int main()
{
    FILE *fp, *fp2;
    int i;
    char s[30];

    fp = fopen("fputceeee.txt", "w");

    fp2 = fopen("fputs.txt", "r");

    if(fp == NULL)
    {
        printf("\n\n FILE NOT FOUND\n\n");
        exit(0);
    }
    else
    {
        printf("\n\n FILE FOUND\n\n");

        fgets(s, 30, fp2);

        for(i=0; i<31; i++)
        {
            fputc(s[i], stdout);
        }
    }

    fclose(fp);
```



```
printf("\n\n");  
return 0;  
}
```

**Output:**

**c) Write a Program that depicts the concept of fgets( ).****Program:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp;
    char s[31][31];
    int i;

    fp = fopen("fputs.txt", "r");

    if(fp == NULL)
    {
        printf("\n\n FILE NOT FOUND\n\n");
        exit(0);
    }
    else
    {
        printf("\n\n FILE FOUND\n\n");
        for(i=0; i<9; i++)
        {
            fgets(s[i], 30, fp);
            fprintf(stdout, "\n\nLine %d is : %s", i+1, s[i]);
        }
    }
    fclose(fp);
    printf("\n\n");
    return 0;
}
```

**Output:**

**d) Write a Program that depicts the concept of fgets( ).**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp;

    fp = fopen("fputs.txt", "w");

    if(fp==NULL)
    {
        printf("\n\n FILE NOT EXIST\n\n");
        exit(0);
    }
    else
    {
        printf("\n\n FILE EXISTS\n\n");
        fputs("Ram is a Good Boy", fp);
    }
    fclose(fp);
    printf("\n\n");
    return 0;
}
```

**Output:**

**e) Write a Program that shows the usage of fread( ) and fwrite( ).**

**Program:**

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

struct stud
{
    int roll;
    char name[100];
};

int main()
{
    FILE *fp;

    int n, i;

    struct stud s[20];

    fp = fopen("djillu.dll","wb");

    if(fp == NULL)
    {
        printf("\n\n FILE NOT FOUND");
        exit(0);
    }
    else
    {
        printf("\n\n FILE FOUND");

        printf("\n\n Enter Number of Students : ");
        scanf("%d", &n);
```

```
printf("\n\n *****Enter %d STUDENTS DATA*****", n);

for(i=0; i<n; i++)
{
    printf("\n\n Enter Student %d Roll : ", i+1);
    scanf("%d", &s[i].roll);

    printf("\n\n Enter Student %d Name : ", i+1);
    scanf("%s", s[i].name);

    fwrite(&s[i], sizeof(struct stud), 1, fp);
}

printf("\n\n Data Wrote to File Successful");

}
fclose(fp);

fp = fopen("djillu.dll", "rb");

if(fp == NULL)
{
    printf("\n\n FILE NOT AVAILABLE");
    exit(0);
}
else
{
    printf("\n\n FILLE FOUND");

    for(i=0; i<n; i++)
    {
        fread(&s[i], sizeof(struct stud), 1, fp);
        printf("\n\n Student %d Roll : %d", i+1, s[i].roll);
        printf("\n\n Student %d Name : %s", i+1, s[i].name);
    }
}
```

```
    }  
    printf("\n\n Read the Data from File Successful");  
}  
fclose(fp);  
  
printf("\n\n");  
return 0;  
}
```

**Output:**

**f) Write a Program that depicts the file random accessing.****Program:**

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

int main()
{
    FILE *fp;
    char c;

    fp = fopen("rrr.txt", "r");

    if(fp == NULL)
    {
        printf("\n\n FILE NOT FOUND");
        exit(0);
    }
    else
    {
        printf("\n\n FILE FOUND");
        while(!feof(fp))
        {
            c = fgetc(fp);
            printf("\n Char %c at Position %ld", c, ftell(fp));
        }

        rewind(fp);

        printf("\n\n Present Position : %ld", ftell(fp));

        fseek(fp, -12, SEEK_END);

        printf("\n\n Position after Seek : %ld", ftell(fp));
    }
}
```

```
        fseek(fp, -5, 1);  
        printf("\n\n Position after Seek_Cur is : %ld", ftell(fp));  
  
    }  
  
    fclose(fp);  
    printf("\n\n");  
    return 0;  
}
```

**Output:**



**g) Write a Program that depicts the concept of Error Handling in C.****Program:**

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

int main()
{
    FILE *fp;

    fp = fopen("dumbukumnar.txt", "r");

    if(fp==NULL)
    {
        perror("Error Read ");
        printf("\n\n Read Error Number : %d", errno);
    }

    else
    {
        fputs("Welcome to PPSC Lab", fp);

        if(ferror(fp))
        {
            perror("Write Error ");
            printf("\n\n Error Number : %d", errno);
        }
    }

    //fclose(fp);
```

```
    printf("\n\n");  
    return 0;  
}
```

**Output:**