FLOAT

10101

\n [21]

%d

101
010

{C}

PROGRAMMING PROJECT;

string

FileName	: C_lab_13/8
Author	: Anjali Kalra
Version	: 1.0

[#########]

/* This file contains all the experiments for C programming! */

\n

Date _____

Expt. No. _____ Page No. _____

Experiment-1

I) Write a C program to print "Hello World"

→ **Aim :** To write a program to print 'Hello World'

→ **Theory :** A basic C- program includes a header file, a main() function and statements ending with a semi colon (;) Here in this experiment we analyze a structure and syntax of a basic C program.

→ **Algorithm :**

① Start
② Include the header file : <stdio.h>
③ Define the main() function.
④ Use the printf() function to print "Hello World"
⑤

→ **Result :** The program was successfully executed to display the desired output i.e 'Hello World'.

Teacher's Signature _____

→ Program :

main.c

Share Run

```c
#include <stdio.h>
int main() {
    printf("Hello World");
    return 0;
}
```

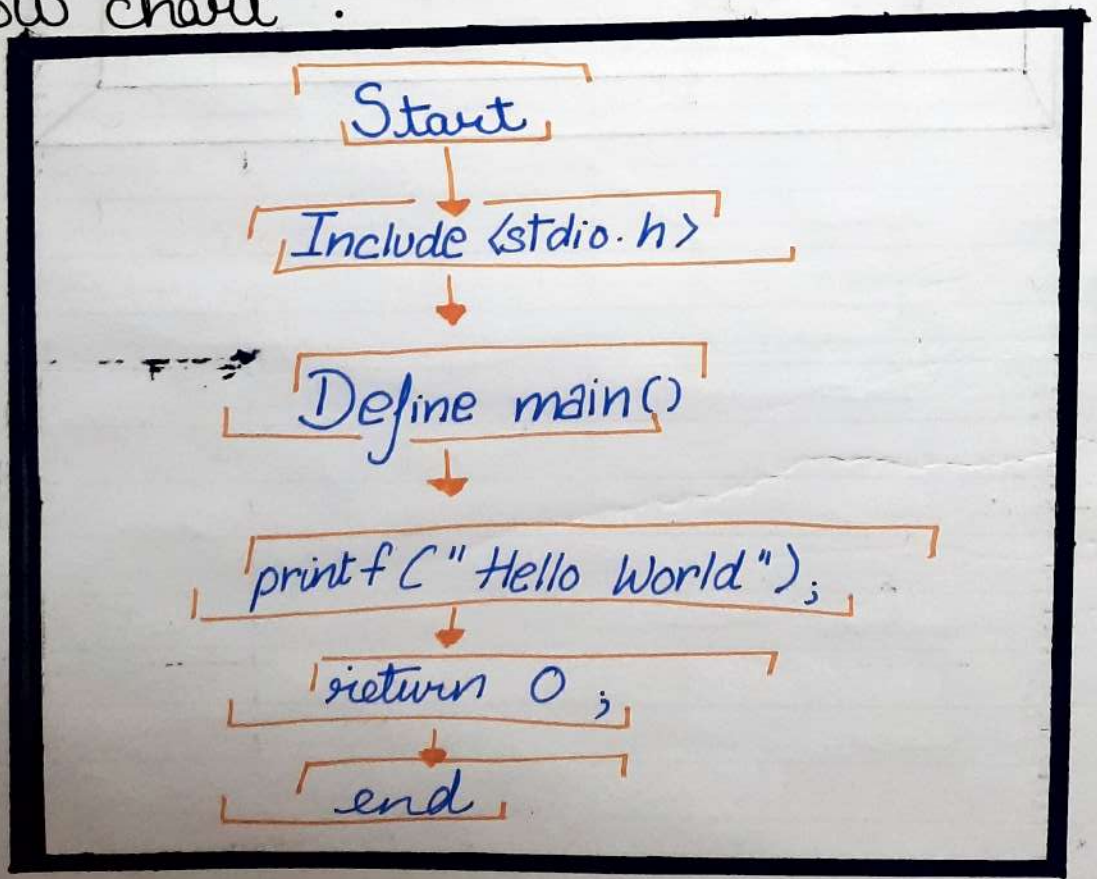
→ Output

Output

```
Hello World

=== Code Execution Successful ===
```

→ Flow chart :



Date _____

Expt. No. _____ Page No. _____

II) Write a C program to print the address in multiple lines (new line).

→ Aim : To write a program to print the output in multiple lines.

→ Theory : In order to get the output printed in multiple line we use \n : escape sequence, it inserts a line break. This experiment demonstrates how to use escape sequences to format output across multiple lines.

→ Algorithm :
① Start
② Include the header file <stdio.h>
③ Define the main() function
④ Use the printf() function with the \n escape sequence to print each line of the address.
⑤ End

→ Result : The program was successfully executed to display the address in multiple lines using the new line escape sequence.

Teacher's Signature _____

→ Program

main.c

Share Run

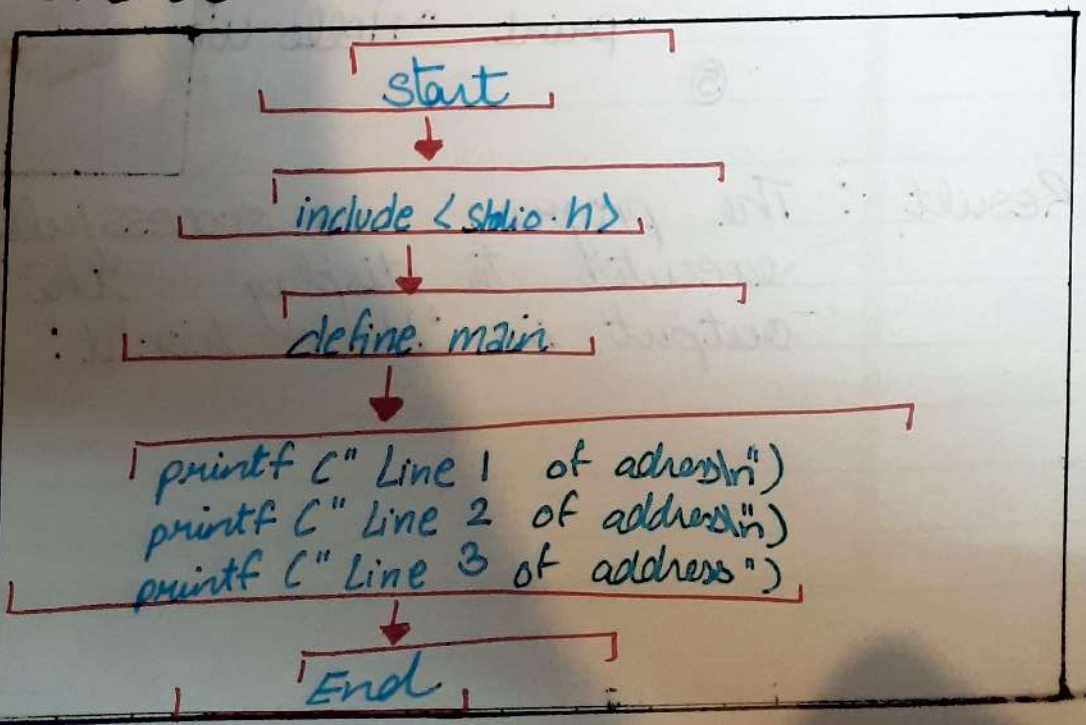
```c
#include <stdio.h>
int main() {
    printf("My name is Anjali kalra.\n");
    printf("I am a student of UPES.\n");
    printf("My major is computer science.\n");
    return 0;
}
```

→ Output

Output

```
My name is Anjali kalra.
I am a student of UPES.
My major is computer science.
```

→ Flow-chart :



Date _____

Expt. No. _____ Page No. _____

III) Write a program that prompts the user to enter their name and age.

→ Aim : To write a program that prompts the user to enter their name and age.

→ Theory : In order write a program that prompts the user to enter their name and age we use "scanf()" function. The %s format specifier reads a string without spaces, while %d reads an integer. The printf() function is used to display output. This experiment demonstrates basic user interaction through input and output functions.

→ Algorithm :
① Start
② Include the header file <stdio.h>
③ Define the main() function.
④ Declare variables to store the name and age.
⑤ Prompt the user to enter their name and read it using scanf().
⑥ Prompt the user to enter their age and read it using scanf()
⑦ Display the entered name and age

Teacher's Signature _____

→ Program :

main.c

Share Run

```c
#include <stdio.h>
int main() {
    int name;
    int age;
    printf("enter ur name:");
    scanf("%s", &name);
    printf("enter ur age:");
    scanf("%d",&age);
    return 0;
}
```

→ Output

Output

```
enter ur name:anjali
enter ur age:17

...Program finished with exit code 0
Press ENTER to exit console.
```

Date _____

Expt. No. _____ Page No. _____

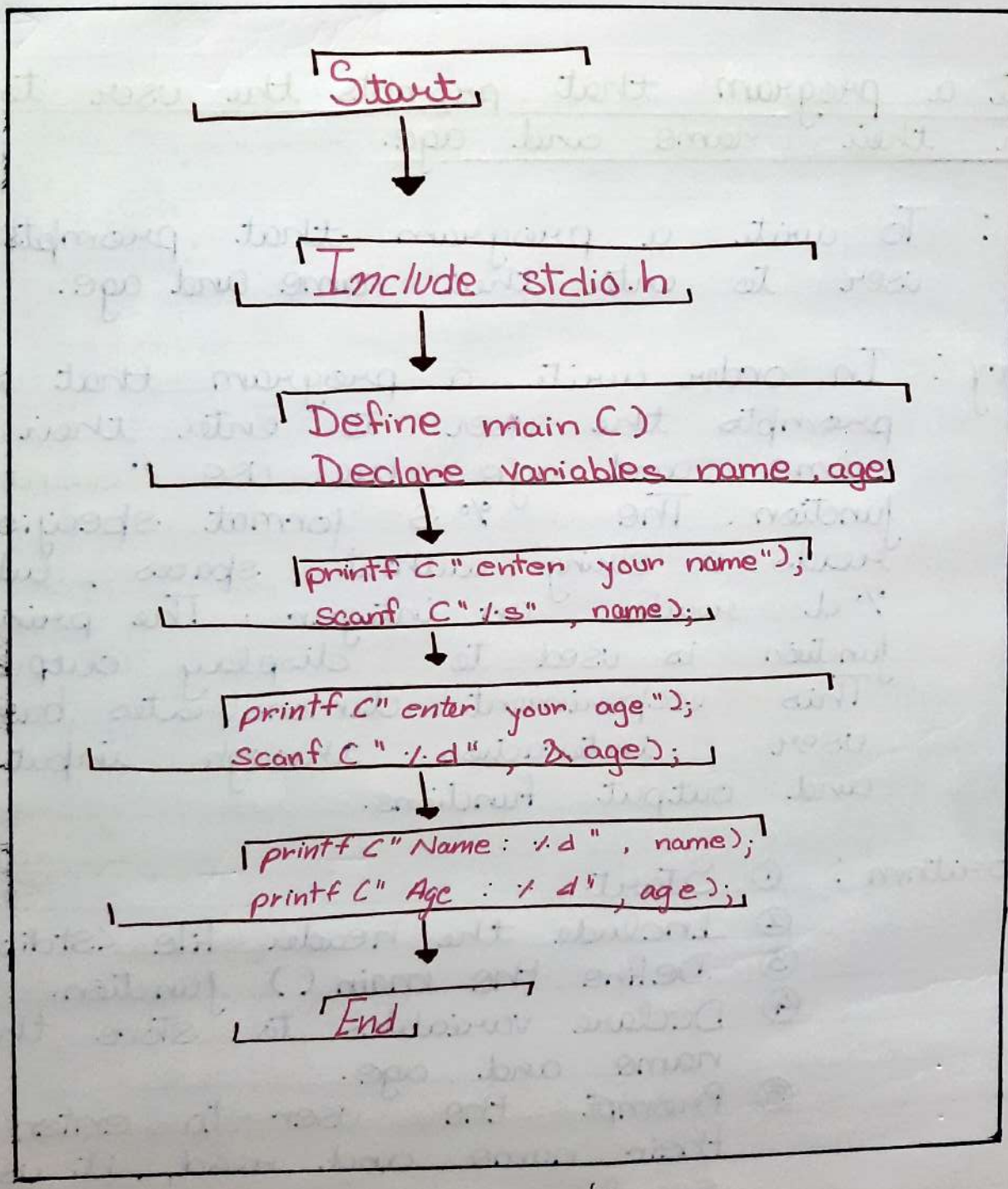
using scanf().

⑧ End

→ Result : The program was successfully executed to accept and display the user's name and age.

O/P -> White

→ Flow-chart :


Start
Include stdio.h
Define main ()
Declare variables name, age
printf ("enter your name");
scanf ("%s", name);
printf ("enter your age");
scanf ("%d", &age);
printf ("Name : %d", name);
printf ("Age : %d", age);
End

Date

Expt. No. Page No.

II) Write a C program to add two numbers, take numbers from user.

→ Aim : To write a program to add 2 numbers, and take the numbers from the user.

→ Theory : In order to print the above code we use scanf() function which can take multiple inputs from the user. The %d format specifier reads integer values. Arithmetic operators such as +, −, *, % allow us to perform basic mathematical operations. This experiment demonstrates reading user input and performing addition.

→ Algorithm : ① Start
② Include the header file <stdio.h>
③ Define the main() function.
④ Declare variables to store 2 numbers and their sum.
⑤ Prompt the user to enter the first number and read it using scanf().
⑥ Prompt the user to enter the second number and read it using

Teacher's Signature

→ Program :

main.c

Share Run

```c
#include <stdio.h>
int main() {
    int number1, number2, sum;
    printf("Enter your First number1:");
    scanf("%d", &number1);
    printf("Enter your Second number2:");
    scanf("%d", &number2);
    sum= number1 + number2;
    printf("Your sum is: %d", sum);
    return 0;
```

→ Output

Output

```
Enter your First number1:6
Enter your Second number2:7
Your sum is: 13
```

Date _____

Expt. No. _____ Page No. _____

scanf().

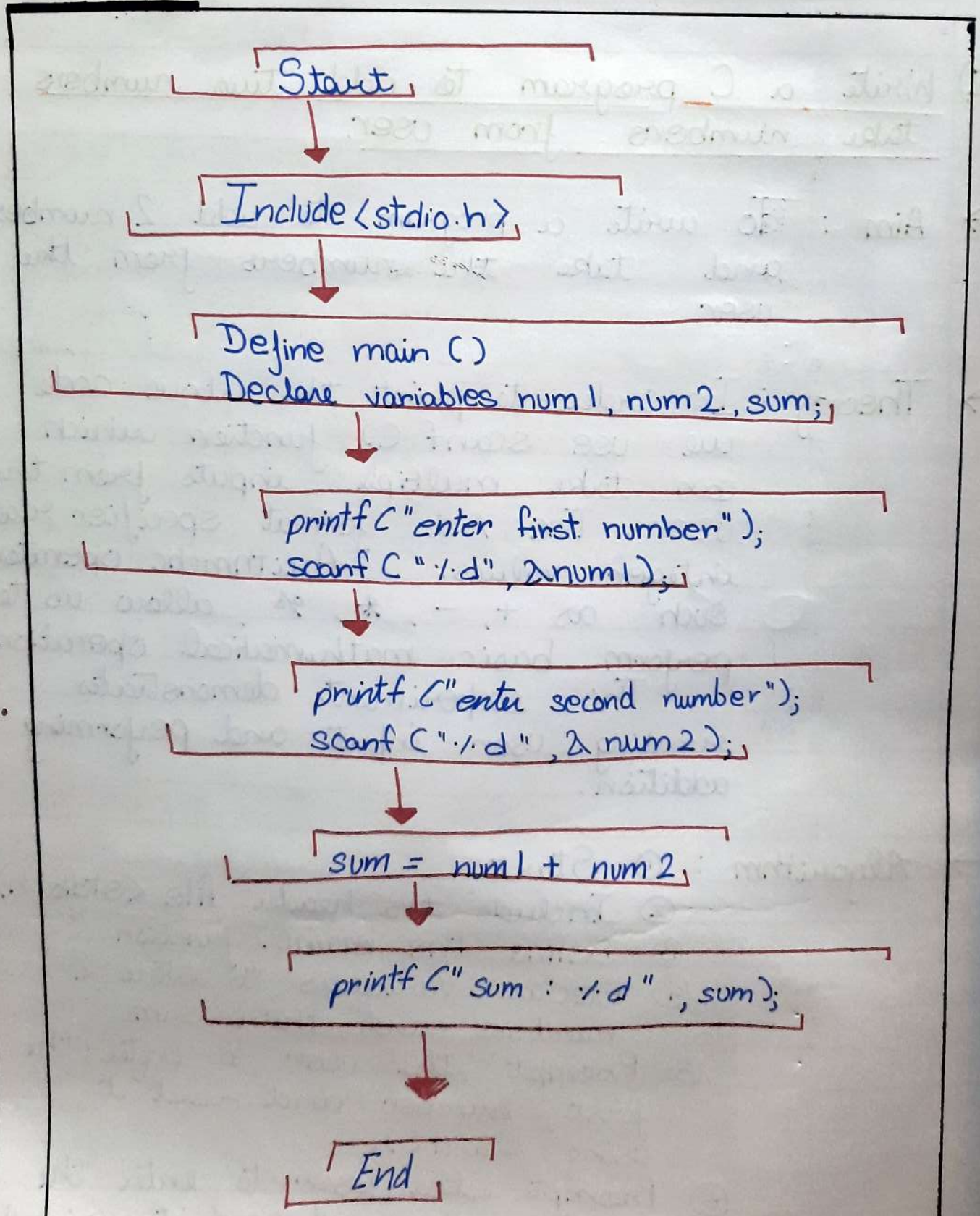
⑦ Calculate the sum of the 2 numbers.

⑧ Display the sum using printf().

⑨ End

→ Result : The program was successfully executed to accept two integer values from the user and print the sum as the output.

→ Flow - chart


Start
Include <stdio.h>
Define main ()
Declare variables num1, num2., sum;
printf("enter first number");
scanf("%d", &num1);
printf("enter second number");
scanf("%d", &num2);
sum = num1 + num2
printf("sum : %d", sum);
End

Date _____

Expt. No. _____ Page No. _____

Experiment-2

OPERATORS

I) WAP a C program to calculate the area and perimeter of a rectangle based on its length and width.

→ Aim : To calculate the area and perimeter of a rectangle based on the length and width.

→ Theory : Here, the user input will be taken using the scanf() function. The formula for the perimeter and area of a rectangle is:

perimeter = 2 X (length + width)
area = length X width

→ Algorithm :
① Start
② Include the header file <stdio.h>
③ Define the main() function
④ Declare variable for length, width, perimeter and area.
⑤ Prompt the user to enter the length and read it using scanf()

Teacher's Signature _____

→ Program :

main.c

Share Run

```c
#include <stdio.h>
int main() {
    float a, b;
    printf("Enter your length:");
    scanf("%f",&a);
    printf("Enter your width:");
    scanf("%f",&b);
    float area, perimeter;
    area= a*b;
    printf("Your area is %.2f\n", area);
    perimeter= 2*(a+b);
    printf("Your perimeter is %.2f", perimeter);
    return 0;
}
```

→ Output :

Output

```
Enter your length:14.2
Enter your width:16.4
Your area is 232.88
Your perimeter is 61.20

=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

⑥ Prompt the user to enter the width and read it using scanf()

⑦ Calculate the perimeter using the formula 2(l + w).

⑧ Calculate the area using the formula (l x w)

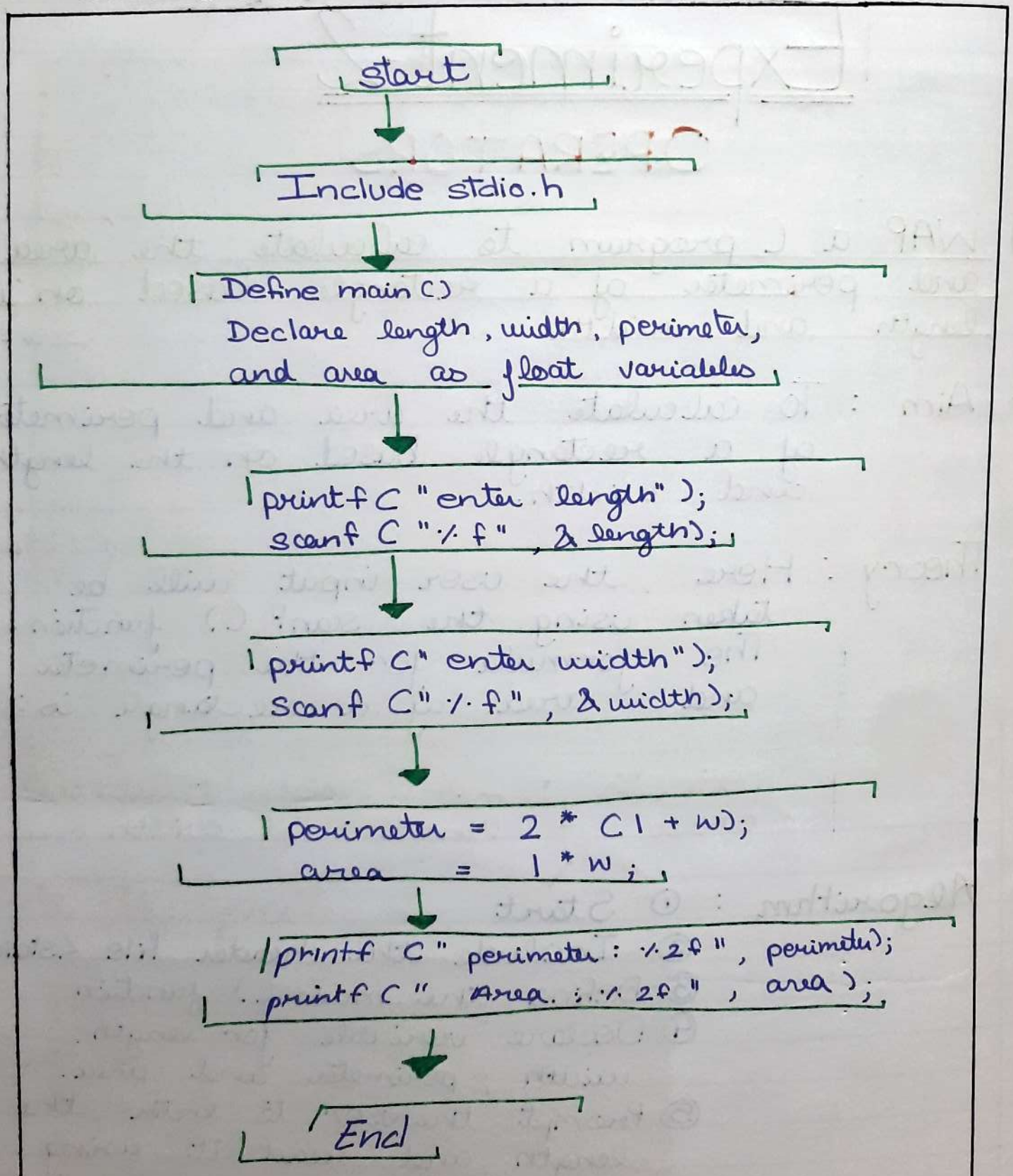
⑨ Display the perimeter and area using printf().

⑩ End

→ Result : The program was successfully executed to calculate and display the perimeter and area of a rectangle based on user input.

Teacher's Signature _____

→ Flow-chart :


start
Include stdio.h
Define main()
Declare length, width, perimeter, and area as float variables
printf("enter length");
scanf("%f", &length);
printf("enter width");
scanf("%f", &width);
perimeter = 2 * (l + w);
area = l * w;
printf("perimeter: %2f", perimeter);
printf("Area: %2f", area);
End

Date _____

Expt. No. _____ Page No. _____

II) WAP a C program to convert temperature from Celsius to Fahrenheit using the formula F = (C * 9/5) + 32.

→ Aim : To convert temperature from Celsius to fahrenheit using the formula F = (C * 9/5) + 32.

→ Theory : Here, the user input is taken using the scanf() function. The formula to convert temperature from Celsius (°C) to Fahrenheit (°F) is :

$$F = \left(C * \frac{9}{5}\right) + 32$$

→ Algorithm :
1. Start
2. Include the header file <stdio.h>.
3. Define the main() function
4. Declare variables for Celsius and Fahrenheit.
5. Prompt the user to enter the temperature in Celsius and read it using scanf()
6. Convert Celsius to Fahrenheit using the formula:

Teacher's Signature _____

→ Program

main.c

Share Run

```c
#include <stdio.h>
int main() {
    float F,C;
    printf("Enter your temperature in Celcius:");
    scanf("%f", &C);
     F= (C*9/5)+32;
    printf("Your temperature in Fahrenheit: %.2f", F);
    return 0;
}
```

→ Output

Output

```
Enter your temperature in Celcius:32.4
Your temperature in Fahrenheit: 90.32

=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

$$F = (C * \frac{9}{5}) + 32$$

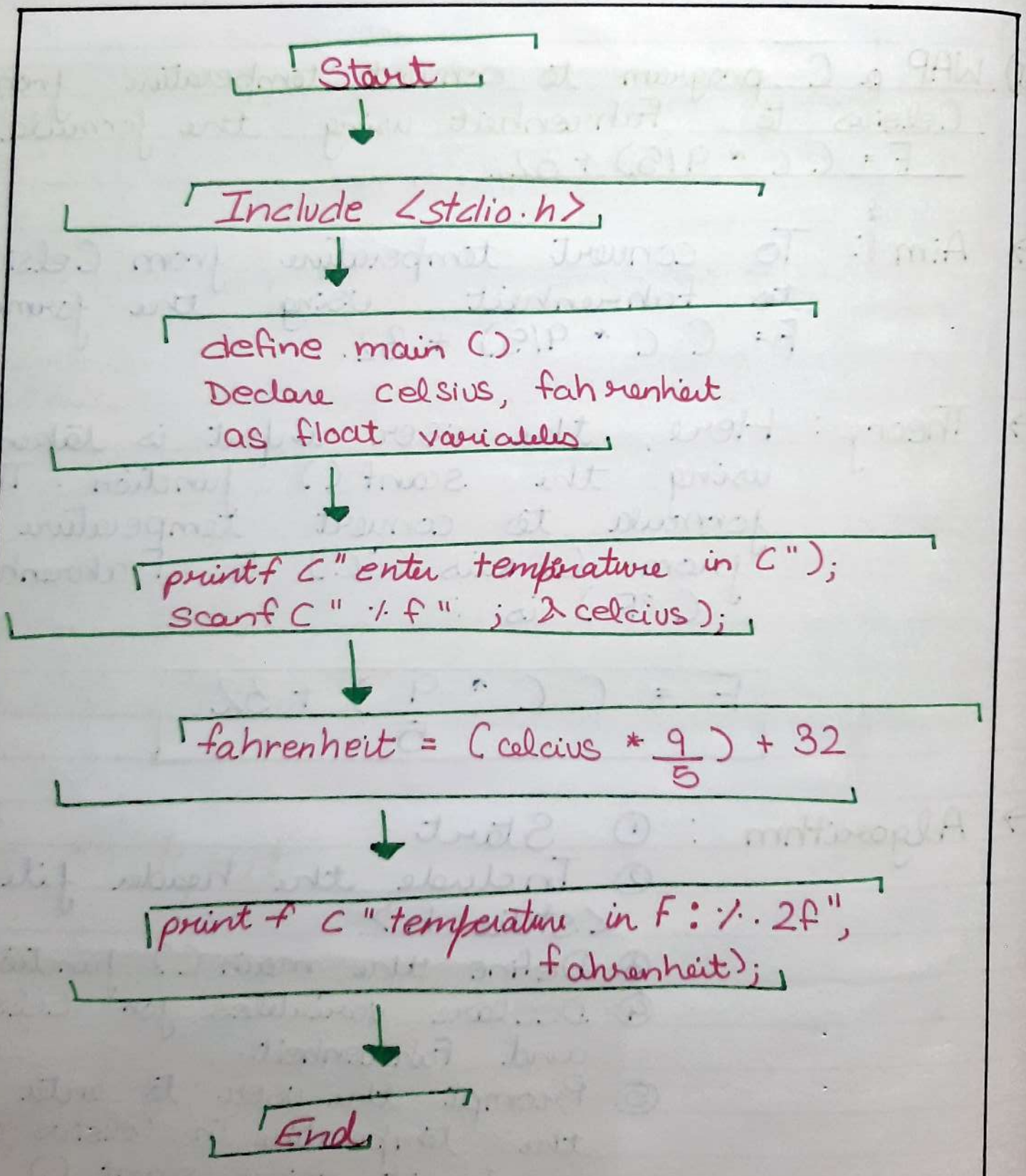
⑦ Display the result using printf().

⑧ End

→ Result : The program was successfully executed to convert a temperature from Celsius to Fahrenheit using the given formula.

Teacher's Signature _____

→ Flow - chart


Start
Include <stdio.h>
define main ()
Declare celsius, fahrenheit
as float variables
printf ("enter temperature in C");
scanf ("%f", &celcius);
fahrenheit = (celcius * 9/5) + 32
printf ("temperature in F: %.2f",
fahrenheit);
End

Date _____

Expt. No. _____ Page No. _____

Experiment - 3

Conditional Statement

I) WAP to check if the triangle is valid or not. If the validity is established do check if the triangle is isoceles, equilateral, right angle or scalene. Take sides of the triangle as input from the user.

→ Aim : To check the validity of a triangle, then interpreting its type, while taking the input from the user.

→ Theory : For any three sides to form a Triangle the Triangle Inequality Theorem must hold true :

$(s_1 + s_2 > s_3)$, $(s_2 + s_3 > s_1)$, $(s_3 + s_1 > s_2)$

Once the validity is confirmed the type of triangle is classified as:

① equilateral - $(a == s_2 == s_3)$

② isosceles - $(s_1 == s_2 \;||\; s_1 == s_3 \;||\; s_2 == s_3)$

③ right angle - $(s_1 + s_2)^2 == s_3^2$

④ scalene - $s_1 != s_2 != s_3$

The program uses conditional statement (if-else)

Teacher's Signature _____

→ Program:

```c
int main () {
    int s1,s2,s3;
    printf("Enter your first side:");
    scanf("%d",&s1);
    printf("Enter your second side:");
    scanf("%d",&s2);
    printf("Enter your third side:");
    scanf("%d",&s3);
    if (s1+s2>s3 && s2+s3>s3 && s3+s1>s2) {
        printf("This triangle exists.\n");
    }
    else {
        printf("Triangle is invaild.");
    }
    if (s1==s2 && s2==s3 && s3==s1) {
    printf("It's an eqilateral triangle.");
    }
    else if (s1==s2 || s1==s3 || s2==s3 ) {
    printf("It's an isoceles triangle.");
    }
    else if ((s1+s2) ^2 == s3^2||(s3+s2) ^2 ==s1^2||(s1+s3) ^2
        ==s2^2) {
        printf("It's a right angled triangle.");
```

```
Enter your first side:12
Enter your second side:9
Enter your third side:6
This triangle exists.
It's a right angled triangle.
```

Output

Date _____

Expt. No. _____ Page No. _____

to check the validity and then classify the triangle accordingly.

→ Algorithum :

① Start
② Input three sides s_1, s_2, s_3.
③ Check validity if,
($s_1 + s_2 > s_3$) && ($s_2 + s_3 > s_1$) && ($s_1 + s_3 > s_2$)
if valid → procood
else → invalid input
④ If valid check type:
- if a == b && b == c → equilateral
- else if a == b || b == c || a == c → isosceles
- else if (a*a + b*b == c*c) → right angled
- else → scalene
⑤ Display result
⑥ Stop

→ Result : The program successfully checks and gives the output that the triangle exists and its a right angle triangle.

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

II) WAP to compute the BMI index of the person and print the BMI values as per the following ranges. You can use the following formula to compute BMI = weight (kgs) / (Height)^2 mts

→ Aim : Write a program that computes the BMI index of the user and prints the value as per the required range, using the formula BMI : $\frac{\text{weight}}{(\text{height})^2}$

→ Theory : The body mass index is a measure that relates a persons weight and height to determine whether the individual is underweight, normal, overweight or obese. The formula for BMI is :

$$BMI = \frac{\text{Weight (Kg)}}{(\text{height})^2}$$

Based on BMI value, the classification is:

- BMI < 15 → starvation
- 15.1 < BMI < 17.5 → anorexic
- 17.6 < BMI < 18.6 → underweight
- 18.7 < BMI < 24.9 → ideal
- 25 < BMI < 25.9 → overweight

Teacher's Signature _____

→ program :

```c
#include <stdio.h>
int main () {
    float w,h;
    printf("Enter your height (in meters):");
    scanf("%f", &h);
    printf("Enter your weight(in kgs):");
    scanf("%f", &w);
    float BMI;
    BMI= w/(h*h);
    printf("Your BMI is %.2f\n", BMI);
    if (BMI<15) {
        printf("Starvation");
    }else if (BMI>=15.1 && BMI<17.5) {
        printf("Anorexic");
    }else if (BMI>=17.6 && BMI<18.5) {
        printf("Underweight");
    } else if (BMI>=18.6 && BMI<24.9) {
        printf("Ideal");
    }  else if (BMI>=25 && BMI<29.5) {
        printf("Overweight");
    }  else if (BMI>=30 && BMI<30.9) {
        printf("Obese");
    }   else {
```

→ output

```
Enter your height (in meters):1.65
Enter your weight(in kgs):81
Your BMI is 29.75
Morbidity obese
```

Date _____

Expt. No. _____ Page No. _____

- $30 < BMI < 39.9$ → obese
- 40 above → morbidity obese

→ Algorithm :
1. Start
2. Input weight (Kgs) and height
3. Compute BMI using the formula.
4. if BMI < 15 : starvation
5. else if BMI < 17.5 → anorexic
6. else if BMI < 18.6 → underweight
7. else if BMI < 24.9 → ideal
8. else if BMI < 25.9 → overweight
9. else if BMI < 30 → obese
10. else → morbidity obese
11. print the result
12. stop

→ Result : The program successfully calculates the BMI of a person using the given formula and categorizes them as underweight, Normal weight, overweight or obese.

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

III) WAP to check if 3 points (x_1, y_1), (x_2, y_2) and (x_3, y_3) are collinear or not.

→ Aim : To write a program to check if 3 points (x_1, y_1), (x_2, y_2), (x_3, y_3) are collinear or not.

→ Theory : Three points are collinear if they lie on the same straight line. The same condition can be checked with the following area formula:

$$\text{Area} : \frac{1}{2}\left[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right]$$

- If the area = 0, then the 3 points are collinear.
- If area ≠ 0, then the 3 points are not collinear.

→ Algorithum :
1. start
2. Input 3 points
3. Compute the area
4. if area == 0 → points are collinear
5. else → points are not collinear
6. Display the result
7. Stop

Teacher's Signature _____

→ program

```c
#include <stdio.h>

int main() {
    int x1, y1, x2, y2, x3, y3;
    printf("coordinates of first point (x1 y1): ");
    scanf("%d %d", &x1, &y1);
    printf("coordinates of second point (x2 y2): ");
    scanf("%d %d", &x2, &y2);
    printf("coordinates of third point (x3 y3): ");
    scanf("%d %d", &x3, &y3);
    int area = x1*(y2- y3)+x2*(y3-y1)+x3*(y1- y2);

    if (area == 0)
        printf("The points are collinear.\n");
    else
        printf("The points are not collinear.");
    return 0;
}
```

→ output

```
oordinates of first point (x1 y1): 3 8
oordinates of second point (x2 y2): 9 6
oordinates of third point (x3 y3): 7 3
he points are not collinear.

=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

→ **Result :** The program successfully determines wheater the given 3 points are coollinear by applying the area of a triangle method.

Teacher's Signature _____

Expt. No. _____ Page No. _____

→ **Result :** The program successfully determines wheater the given 3 points are coollinear by applying the area of a triangle method.

Date _____

Expt. No. _____ Page No. _____

IV) According to the gregorian calender, it was a Monday on the date 01/01/01. If any year is input through the keyboard, write a program to find out what is the day on the 1st January of this year

Aim : Write a program that give any year as input, find out the day of the week on 1st January was monday.

Theory : ① The gregorian calendar follows a 7-day week cycle
- monday
- tuesday
- wednesday
- Thursday
- Friday
- Saturday
- Sunday

② 01/01/01 was monday

③ every normal year has 365 days i.e. 365

④ every leap year has 366 days ie 366

So, to find the day on 01/01/01 of given year:

i) count total days from 01/01/01 to 01/01/(given year)

Teacher's Signature _____

→ program :

```c
#include <stdio.h>

int main() {
    int year, i, days = 0, day;
    printf("Enter year: ");
    scanf("%d", &year);
    for (i = 1; i < year; i++) {
        if ((i % 400 == 0) || (i % 4 == 0 && i % 100 != 0))
            days = days + 366;
        else
            days = days + 365;
    }
    day = (days % 7 [illegible];

    printf("On 01/01/%d it was: ", year);
    switch(day) {
        case 0: printf("Sunday\n"); break;
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3: printf("Wednesday\n"); break;
        case 4: printf("Thursday\n"); break;
        case 5: printf("Friday\n"); break;
        case 6: printf("Saturday\n"); break;
```

→ output

```
Enter year: 1997
On 01/01/1997 it was: Wednesday


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

② Take remainder of days ÷ 7.
③ map remainder to a weekday.

A leap year is one that :

① Is divisible by 400, OR
② Divisible by 4 but not by 100.

→ Algorithm :
① Start
② Input the year (y)
③ Initialize (days = 0)
④ Loop from year 1 to (y-1):
- if year is leap → add 366 to days
- else → add 365 to days

⑤ Compute days % 7
⑥ Since 01/01/01 was monday, map remainder to day :
- 0 → monday, 1 → tuesday ....

⑦ print result
⑧ Stop

→ Result : The program successfully computes the day of the week on 1st Jan of any given year using the gregorian calender rules.

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

V) WAP using ternary operator, the user should input the length and breath of a rectangle, one has to find out which rectangle has the highest perimeter. The minimum number of rectangles should be 3.

→ Aim: To write a program using the ternary operator where the user inputs the length and breath of 3 rectangles and the program finds out which rectangle has the highest perimeter

→ Theory: The perimeter of a rectangle is:

$$P = 2 \times (length + breath)$$

- The ternary operator in C, is to a compact way to write conditional statements.
- Take input for 3 rectangles (length & breath)
- Calculate perimeter of each.
- Use nested ternary operators to compare and find the rectangle with the maximum perimeter.

→ Algorithm: ① Start
② Input length and breath
③ compute perimeter

Teacher's Signature _____

→ program :

```c
#include <stdio.h>

int main() {
    int x1, y1, x2, y2, x3, y3;
    printf("Enter the length and breath of first triangle:");
    scanf("%d %d", &x1,&y1);
    printf("Enter the length and breath of second triangle:");
    scanf("%d %d", &x2,&y2);
    printf("Enter the length and breath of third triangle:");
    scanf("%d %d", &x3,&y3);
    int p1,p2,p3,max;
    p1= 2*(x1+y1);
    p2= 2*(x2+y2);
    p3= 2*(x3+y3);
    max=(p1>p2)?((p1>p3)? p1:p3): ((p2>p3)? p2:p3);
    printf("The triangle with greatest perimeter:%d\n", max);
    return 0;
}
```

→ output

Enter the length and breath of ji

```
Enter the length and breath of first triangle:7 8
Enter the length and breath of second triangle:5 9
Enter the length and breath of third triangle:6 5
The triangle with greatest perimeter:30
```

Date _____

Expt. No. _____ Page No. _____

④ Compare perimeter using ternary operator
⑤ print which rectangle has the highest perimeter.
⑥ Stop

→ Result : The program was successfully executed using the ternary operator to determine which rectangle has the highest perimeter among three user entered rectangles.

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

Experiment - 11

Bitwise Operator

I) Write a program to apply bitwise OR, AND and NOT operator on bit level.

→ Aim : To write a program that applys bitwise OR, AND and NOT.

→ Theory : Bitwise operators directly operate on the binary representation of integers.
main operators :

① AND (&) : compares each bit and returns (1) if both bits are 1, else 0.

② OR (|) : compares each bit and returns (1) if atleast one bit is (1)

③ NOT (~) : inverts each bit (changes 0 to 1 and 1 to 0).

→ Algorithm : ① Start
② input 2 integers (a and b)
③ perform AND : a & b
④ perform OR : a | b

Teacher's Signature _____

→ program

```c
#include <stdio.h>

int main() {
    int a=5;
    int b=6;
    printf("bitwise a or b is %d\n", a|b);
    printf("bitwise a and b is %d\n", a&b);
    printf("bitwise a not b is %d\n", ~a);
    return 0;
}

```

→ output

```
bitwise a or b is 7
bitwise a and b is 4
bitwise a not b is -6


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

⑤ perform NOT : ~a , ~b
⑥ Display results
⑦ Stop

→ **Result:** The program was successfully executed to perform bitwise AND, OR and NOT operations on integers at the bit level.

Date _____

Expt. No. _____ Page No. _____

II) Write a program to apply left shift and right shift operator.

→ Aim : To write a program to apply left shift and right shift operator.

→ Theory : Shift operator are used to move bits of a number to the left and right.

- Left shift (<<) : shifts bits to the left inserting 0's on the right.
- Right shift (>>) : shifts bits to the right inserting / discarding rightmost bits.
- For left shift : multiplying no's by 2^n.
- For right shift : dividing no's by 2^n.

→ Algorithm :
1. Start
2. Input an integer a and the number of shifts n.
3. Compute a<<n (left shift)
4. Compute a >> n (right shift)
5. Display results
6. Stop

→ Result : The program was successfully

Teacher's Signature _____

→ program

```c
#include <stdio.h>

int main() {
    int num= 5;
    int leftshift,rightshift;
    leftshift = num << 1;
    rightshift = num >> 1;
    printf("Our number: %d", num);
    printf("\nleft shift (num << 1): %d", leftshift);
    printf("\nright shift (num >> 1): %d\n", rightshift);

    return 0;
}

```

→ output

```
Our number: 5
left shift (num << 1): 10
right shift (num >> 1): 2


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

Experiment - 3.2

Loops

① WAP to enter numbers till the user wants. At the end, it should display the count of positive, negative and zeros entered.

→ Aim : To write a C program to enter numbers till the user wants. At the end, the program should display the count of positive, negative and zero's entered.

→ Theory : This code will be executed using for loop.

① In this program the user first enters how many numbers they want to input.

② For each number
- if num > 0 : positive
- if num < 0 : negative
- if num = 0 : 0

③ Three variables (pos, neg, zero) are created.

④ At the end the program displays the count.

Teacher's Signature _____

Code :

```
#include <stdio.h>
int main() {
int n, num;
int pos, neg, zero;
printf("How many numbers do you want to enter?");
scanf("%d", &n);

for (int i = 1 ; i<=n ; i++) {
printf("Enter number %d ", i);
scanf("%d ", &num);

if (num>0) {
 pos++;
}
else if (num<0) {
 neg++;
}
else {
 zero++;
}
printf("Count of positive numbers: %d\n");
printf("Count of negative numbers: %d\n", neg);
printf("Count of negative numbers: %d\n" zero);

return 0;

}
```

Date _____

Expt. No. _____ Page No. _____

② WAP to print the multiplication table of the numbers entered by the user. It should be in the correct formatting.
Num * 1 = num.

→ Aim : To write a C program to print the multiplication table of a number entered by the user in proper format

→ Theory : A multiplication table displays the product of a number with a sequence of integers (commonly 1 - 10).
We can use - for loop to perform this repetitive task.
By using loops we multiply the given number by all integers from 1-10 and display each result in a structured format.

→ Algorithum :
1) start the program
2) declare num and i
3) read a number from the user and store in num.
4) Use a 'for loop' that runs from i = 1 to i <= 10.
5) Inside the loop, calculate num * i
6) Display the result : num * i = product
7) End the program.

Teacher's Signature _____

→ Code

```c
#include <stdio.h>
int main() {
    int num, i;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Multiplication Table of %d\n", num);
    for (i = 1; i <= 10; i++) {
      printf("%d * %d = %d\n", num, i, num * i);
    }
    return 0;
}

```

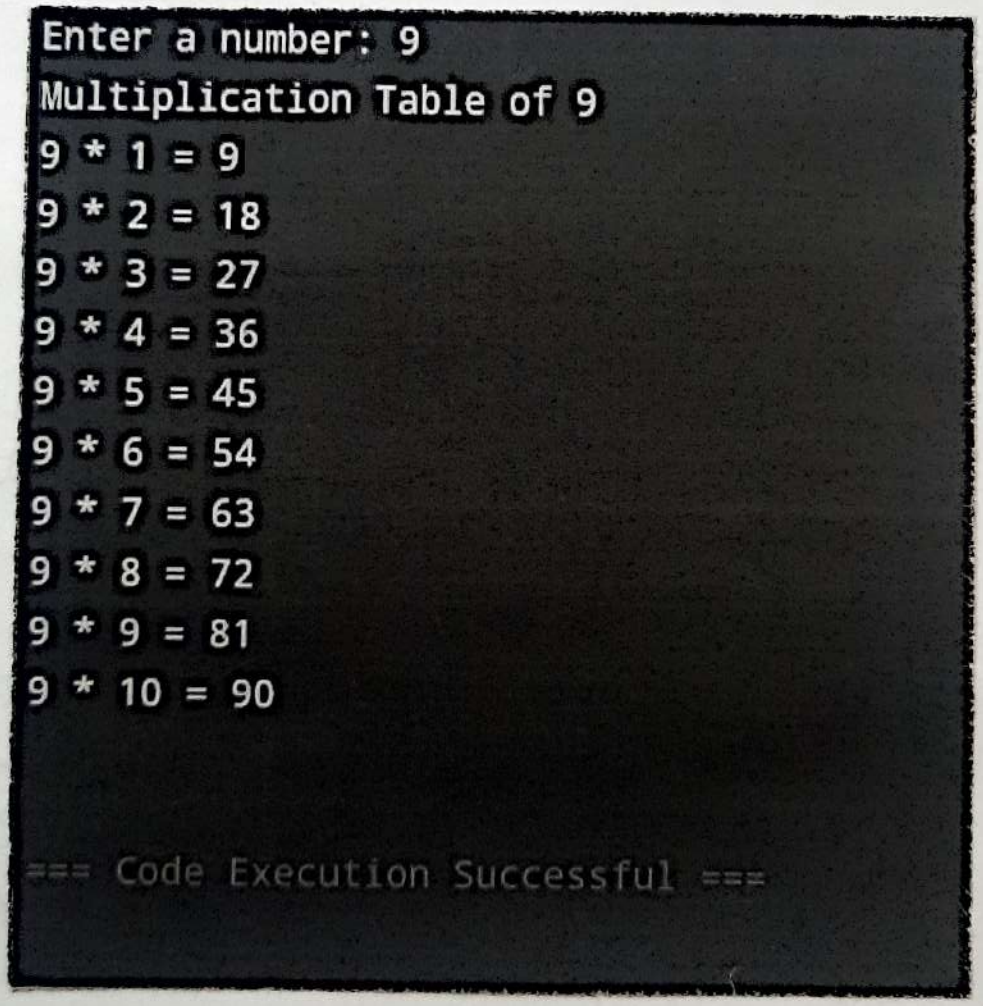
Date _____

Expt. No. _____ Page No. _____

→ Result : The program successfully prints the multiplication table of the numbers entered by the user in the correct format.

Teacher's Signature _____

→ output :


Enter a number: 9
Multiplication Table of 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
=== Code Execution Successful ===

Date _____

Expt. No. _____ Page No. _____

③ WAP to generate the following set of output

a)
```
1
2 3
4 5 6
```

b)
```
      1
    1   1
  1   2   1
1   3   3   1
1 4   6   4   1
```

→ Aim: to write a program that gives out the given set of output.

→ Theory: The theory for both the sets will be same i.e nested loops are used to print this pattern. The outer loops control the numbers of rows, while the inner loop controls the number of elements printed in each row.

→ Algorithum: a)

1. Start the program
2. Declare the integer variable i, j and num=1
3. Use an outer loop for rows (1-3)
4. Inside the outer loop, use an inner loop to print numbers in each row
5. After each row, move to next line
6. End the program

Teacher's Signature _____

a)

Code :

```c
#include <stdio.h>
int main() {
    int i, j, num = 1;
    for (i = 1; i <= 3; i++)
    {
        for (j = 1; j <= i; j++)
        {
            printf("%d ", num);
            num++;
        }
        printf("\n");
    }
    return 0;
}

```

output :

```
1
2 3
4 5 6


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

b) 1) start the program
2) Input the number of rows from the user
3) use an outer loop for rows (i=0 to n-1)
4) use spaces to center the number
5) Initialize num=1 at the start of each row.
6) Use an inner loop to print elements of current row using the relation:

$$num = num \times \frac{(i - j)}{(j + 1)}$$

7) print the numbers in the correct format.
8) move to the next line after each row.
9) End the program.

→ Result : a) The program has successfully generated desired pattern using nested loops in C.

b) The program has successfully generated Pascal's triangle up to the number of rows entered by the user.

Teacher's Signature _____

b) code :

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

→ output :

```
Enter number of rows: 4
    1
   1 1
  1 2 1
 1 3 3 1


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

⑤ Ramanujan Number is the smallest number that can be expressed as the sum of 2 cubes in 2 different ways. WAP to print all such numbers up to a reasonable limit.

Ex: Ramanujan no : 1729

→ Aim: To write a C program to find and print all Ramanujan numbers up to a given limit.

→ Theory: A Ramanujan number is the smallest number that can be expressed as the sum of 2 cubes in 2 different ways.

Eg: $1729 : 12^3 + 1^3 = 10^3 + 9^3$

- Compute all possible sums of cubes a^3+b^3 where a, b are integers.
- Check if any number can be represented by more than one distinct pair (a, b)

→ Algorithm:

① Start the program
② Read the upper limit
③ Loop for all integers a and b
$a \leq b$
④ Compute sum = $a^3 + b^3$
⑤ if the same sum is obtained for 2

Teacher's Signature _____

code:

```
#include <stdio.h>
    int limit = 2000;
    printf("Ramanujan Numbers upto %d are:\n", limit);
    for(int num=1; num<=limit ; num++) {
    for(int a=1 ; a*a*a < num; a++) {
    for(int b=a+1; b*b*b < num; b++) {
    for(int d = c+1; d* d *d < num; d++) {
     if((a*a*a + b*b*b == num) &&
        (c* c*c + d*d*d == num) &&
        !((a==c) && (b==d))) {
        printf("%d = %d^3 + %d^3 = %d^3 + %d^3\n",
                num, a, b, c, d);
    }
  }
 }
}
```

output:

Ramanujan Numbers upto 2000 are:

1729 = 1^3 + 12^3 = 9^3 + 10^3

1729 = 9^3 + 10^3 = 12^3 + 1^3

Date _____

Expt. No. _____ Page No. _____

Experiment - 4

Variable and scope of variable

① Declare a global variable outside all functions and use it inside various functions to understand its accessibility.

→ Aim: To write a C program to declare a global variable outside the functions and use it inside various functions to understand its accessibility.

→ Theory: Variables can be classified on their scope and lifetime.
A global variable is declared outside all the functions, typically at the top of the program. It is accessible by all functions in the program and retains its value throughout the program's execution.

→ Algorithum:
① Start the program
② Declare global variable outside all the functions

Teacher's Signature _____

Code:

```
#include <stdio.h>
int globalVar = 10;

void display();
void modify();

int main() {
printf("Inside main(): globalVar = %d\n", globalVar);
display();
modify();
printf("Back to main(): globalVar = %d\n", globalVar);

return 0;
}
void display() {
printf("Inside display(): globalVar = %d\n", globalVar);
}
void modify(){
printf("Inside display(): globalVar = %d\n", globalVar);
}

void modify() {
globalVar = 15;
printf("Inside modify(): globalVar = %d\n", globalVar);
}
```

Date _____

Expt. No. _____ Page No. _____

③ Define 2 or more functions.
④ Inside each function use or modify the global variable.
⑤ Print the global variable in different functions to observe its behavior.
⑥ End the program.

→ Result : The program demonstrates that a global variable declared outside all functions can be accessed and modified by any function in the program. Its value persists throughout the program's execution.

Teacher's Signature _____

Output

Initial Value of global variable : 10

Value of global variable inside display : 10

Value of global variable inside modify : 15

Value of global variable after modify : 15

Output

Inside main() : global Var = 10

Inside display() : global Var = 10

Inside modify() : global Var = 15

Back to main: global Var = 15

Date _____

Expt. No. _____ Page No. _____

② Declare a local variable inside a function and try to access it outside the function. Compare this with accessing the global variable from within the function.

→ Aim : To write a C program to demonstrate the difference between local and global variable by declaring a local variable inside a function and trying to access it outside the function.

→ Theory : A local variable is declared inside a function or a block and can only be accessed within that function or block.
Its scope is limited to that function and its created when the function is started and destroyed when it ends.

→ Algorithm : ① Start the program
② Declare global variable outside the function.
③ Declare local variable inside one of the functions
④ Access the global variable inside a function.

Teacher's Signature _____

Code :

```
#Include <stdio.h>
int GlobalVar = 50;
void test Function ();
int main () {
    int local main = 10;
        printf (" Inside main(): GlobalVar = %d\n", globalVar);
        printf (" Inside main(): local main = %d \n", local main);

    test function ();

    return 0;

}

void test Function () {
    int local test = 20;
    printf (" Inside testFunction () : globalVar = %d\n", globalVar);
    printf (" Inside test Function() : local test %d\n", local Var);

}
```

output :

```
Inside main () :  globalVar = 50
Inside main () :  localMain = 10
Inside test Function(): global Var = 50
Inside test Function(): localTest = 20
```

Date _____

Expt. No. _____ Page No. _____

5. Access local variable outside the function – it will cause an error.
6. End of program.

→ Result :
- The local variable can only be accessed within the function where it is declared.
- The global variable can be accessed and modified from any function in the program.
- Attempting to access a local variable outside its scope results in an error.

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

③ Declare variables within different code blocks and test their accessibility within and outside those blocks.

→ Aim : To write a program to declare variables within different code blocks and test their accessibility within and outside those blocks.

→ Theory : In C, a block is a section of code enclosed within curly braces. A variable declared inside a block has block scope, meaning it can be accessed within that block. When the program execution leaves the block, the variable is destroyed and no longer accessible. Variables declared in the outer block are accessible by inner blocks, but not vice versa.

→ Algorithm :
① Start the program.
② Declare the variable in the outermost block (inside main).
③ Create an inner block using { } and declare another variable.
④ Print both variables inside the inner block.
⑤ Try to print the inner block variables

Teacher's Signature _____

→ code :

```c
#include <stdio.h>
int main()
{
    int x = 10;
    printf("Outside inner block: x = %d\n", x);
    {
        int y = 20;
        printf("Inside inner block: x = %d\n", x);
        printf("Inside inner block: y = %d\n", y);
    }
    printf("Outside inner block again: x = %d\n", x);
    return 0;
}

```

Date _____

Expt. No. _____ Page No. _____

outside its block.

6. Observe which variables are accessible and where.
7. End the program.

→ **Result** : The program demonstrates that:

① Variables declared inside a block are accessible only within that block.

② Variables declared in an outer block are accessible inside nested blocks.

③ Once the block ends, its variable are destroyed and cannot be accessed further.

Teacher's Signature _____

→ output

```
Outside inner block: x = 10
Inside inner block: x = 10
Inside inner block: y = 20
Outside inner block again: x = 10


=== Code Execution Successful ===
```

Date _____

Expt. No. _____ Page No. _____

④ Declare a static local variable inside a function. Observe how its value persists across function calls.

→ Aim : To write a C program to declare a static local variable inside a function and observe how its value persists across multiple function call.

→ Theory : A static local variable is declared inside a function using the keyword 'static'. It retains its value between multiple calls to the same function. It also has local scope (accessible only inside the function) but global lifetime (exists for the entire duration of the program). Each time the function is called, the static variable is not re-initialized, it keeps the previous value.

→ Algorithm : ① Start the program.
② Create a function that declares a static local variable and increments its each time the function is called.
③ Call this function multiple times from main.
④ Observe how static variable retains its value between calls.

Teacher's Signature _____

→ code

```c
#include <stdio.h>
void myFunction() {
    static int count = 0;
    count++;
    printf("myFunction() has been called %d time\n",count);
}
int main() {
    myFunction();
    myFunction();
    myFunction();
    return 0;
}

```

⑤ End the program

→ Result : The program successfully demonstrates that a static local variable inside a function retains its value across multiple function calls, even though its scope is limited to that function.

→ output

```
myFunction() has been called 1 time
myFunction() has been called 2 time
myFunction() has been called 3 time


=== Code Execution Successful ===
```

Experiment - 5

Arrays

① WAP to read a list of integers and store it in a single dimensional array. Write a C program to print the second largest integer in a list of integers.

→ Aim : To write a C program to read elements into a single-dimensional array and find the largest element among them.

→ Theory : An array is a collection of elements of the same data type stored in contiguous memory locations. Each element element in the array can be accessed using an index.

1) Initialize max with the first element of the array.
2) If any element is greater than the current max, update max with that element.
3) After completing the loop, max holds the 2nd greatest element.

→ Algorithm : ① Start the program.
② Declare an array x[5] and integer variables i and max.
③ Use a for loop to input 5 elements

into the array.

④ Initialize max = x[0].

⑤ Use another for loop to transverse the array:
→ if x[i] > max,
assign max = x[i].

⑥ Print the 2nd greatest element.

⑦ End the program

→ **Result** : The program successfully displays the 2nd greatest element out of 5 in the given array.

Code

```
#include <stdio.h>
int main () {
int n, i, a[100];
int max1, max2;
printf("Enter number of elements: ");
scanf("%d", &n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
max1= max2 = -9999;
for(i=0; i<n; i++) {
 if(a[i] > max1) {
   max 2 = max 1;
   max 1 = a[i];
 }
 else if(a[i] > max && a[i] != max1) {
     max2 = a[i];
 }
}
printf("Second largest =%d", max2);
return 0;
}
```

→ output :

enter element in array = 88
enter element in array = 76
enter element in array = 48
enter element in array = 92
enter element in array = 44

2nd greatest element : 76

② WAP to read a list of integers and store it in a single dimensional array. Write a C program to count and display positive, negative odd and even numbers in an array.

→ Aim: To write a C program to read a list of integers, store them in an array, and count the number of positive, negative, odd, even numbers.

→ Theory: An array is a collection of elements of the same data type stored in consecutive memory locations.
We use looping and conditional statements to check each element of an array and classify as:

① pos : greater than 0
② neg : less than 0
③ even : divisible by 2
④ odd : not divisible by 2

→ Algorithm: ① Start the program.
② Declare an integer array a[50] and variables for counters pos, neg, even, odd
③ Use a for loop to input all array element.
④ Initialize all counters to 0.
⑤ Transverse the array using another loop
- the number is >0, increment pos
- the number is <0 increment neg
- the number divisible by 2, increment even

Code :

```
#include <stdio.h>
int main () {
 int n, a[100], i;
 int  pos =0, neg=0, odd=0, even=0;
  printf ("Enter the numbers of elements: ");
  scanf ("%d", &n);
  for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
  for (i=0; i<n; i++) {
   if (a[i] >0) pos++;
   if (a[i] <0) neg++;
   if (a[i] %2 ==0) even++;
    else odd++;
  }
  printf ("positive= %d Negative = %d odd = %d Even = %d",
            pos, neg, odd, even)

    return 0;
}
```

→ output:

```
Enter number of elements: 4
Entu 4 integers:
 7
 0
 6
-9
positive numbers : 2
negative numbers: 1
even numbers : 2
odd number : 2
```

, otherwise , increment odd.

⑥ Display the counts of pos, neg , odd and even numbers.

⑦ Stop the program.

→ **Result** : The program successfully reads the integers into an array and correctly counts and displays the number of pos, neg , odd and even numbers.

③ WAP to read a list of integers and store it in a single dimensional array. Write a C program to find the frequency of a particular number in a list of integers.

→ Aim : To write a C program to read a list of integers, store them in a single-dimensional array, and find the frequency (number of occurrences) of a particular number.

→ Theory : An array in C is a collection of the same type stored in continuous memory location.
In this program, an integer array is used to store the elements entered by the user.
We then input a specific number and check how many times it appears in the array.

→ Algorithm: ① Start the program
② Declare an integer array a[50], integer variables n, i, num and count = 0.
③ Ask the user to enter the no of elements n.
④ Input all elements of the array using a loop.
⑤ Initialize count = 0
⑥ if a[i] = num, increment count.
⑦ After the loop ends, display frequency of

Code:

```
#include <stdio.h>
int main() {
int n, a[100], num, i, freq=0;
printf("Enter number of elements:");
scanf("%d", &n);

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

printf("Enter number to find frequency:");
scanf("%d", &num);

for(i=0; i<n; i++)
 if(a[i]==num)
   freq++;

printf("Frequency of %d = %d", num, freq);
 return 0;
}
```

output:

```
Enter no of elements: 3
Enter 3 digits:
6
8
9
Enter the no to find its frequency: 8
Frequency of 8: 2
```

num.

⑧ End the program.

→ **Result:** The program has successfully read a list of integers into an array and finds the frequency of the specified number.

code :

```
#include <stdio.h>
int main() {
int A[10][10], B[10][10], C[10][10];
int m, n, p, q, i, j, k;

printf("Enter no of rows and columns of matrix A");
scanf("%d %d", &m, &n);

printf("Enter rows and columns of matrix B");
scanf("%d %d", &p, &q);

if(n != p) {
printf("matrix multiplication not possible");
return 0;
}
printf("Enter elements of matrix A");
for(i=0; i<m; i++)
for(j=0; j<n; j++)
scanf("%d", &A[i][j]);
printf("Enter elements of matrix B");
for(i=0; i<p; i++)
for(j=0; j<q; j++)
scanf("%d", &B[i][j]);

for(i=0; i<m; i++) {
for(j=0; j<q; j++) {
C[i][j] = 0;
for(k=0; k<n; k++)
C[i][j] = C[i][j] + A[i][k] * B[k][j];
}
}

printf("Resultant matrix : ");
for(i=0; i<m; i++) {
for(j=0; j<q; j++)
printf("%d", C[i][j]);
printf("\n");
}
return 0;
}
```

④ Write a program that reads two matrices (A x B) and B (p x q) and computes the product A and B. Read matrix A and matrix B in row major order resp. Print both the input and matrices and resultant matrix with suitable heading and output should be in matrix format only. Program must check the compatibility of orders of the matrices for multiplication. Report appropriate message in case of incompatibility.

→ Aim : To write a C program that reads 2 matrices A (m x n) and B (p x q), checks if they can be multiplied, and if compatible, computes and displays their product in matrix form.

→ Theory : Matrix multiplication is a binary operation that produces a matrix from 2 matrices. For 2 matrices A (m x n) and B (p x q) to be multiplied, the number of columns in A must be equal to the number of rows in B (n = p)

This program reads both matrices in row-major order, performs the compatibility check, and displays all

output:

Enter order of matrix A(m,n) = 2, 3

Enter elements:

1 2 3
4 5 6

Enter order of B matrix = 3, 2

Enter elements

7 8
9 10
11 12

Resultant :

58 64
139 154

→ Algorithm :

① Start the program.

② Declare matrices A[10][10], B[10][10] and C[10][10] and integers m, n, p, q, i, j, k.

③ Input the order (rows and columns) of matrix A
→ m and n.

④ Input the elements matrix A.

⑤ Input the order of matrix B → p and q.

⑥ Input the elements of a matrix B.

⑦ Check if the matrices are compatible for multiplication.
→ if n!=p, print "matrix multiplication not possible" and stop the program.

⑧ Otherwise, compute the product :
→ for each i from 0 to m-1
→ for each j from 0 to q-1
→ initialize c[i][j] = 0
→ for each k from 0 to n-1, compute c[i][j] = A[i][k] * B[k][j]

⑨ Print matrices A, B and C in matrix format.

⑩ Stop the program.

→ Result : The program successfully multiplies 2 matrices (if compatible) and displays the input and resultant matrices in proper format.
If matrices are incompatible for multiplication, it displays an appropriate message.

Anjali Kalra , 590024902 , B-20

CLASSMATE
Date :
Page :

Experiment-6

Functions

① Develop a recursive and non recursive function FACT (num) to find the factorical of a number, n! defined by Fact(n) = 1, if n=0. Otherwise, FACT(n) = n * FACT(n-1). Using this function, write a C program to compute the binomial coefficient. Tabulate the results for different values of n and r with suitable messages.

→ Aim : To develop recursive and non-recursive functions FACT(num) to calculate the factorial of a number and use these functions to compute the binomial coefficient.

→ Theory : The factorial of non-negative integer n, denoted by n!, is defined as,

- $n! = 1$, if $n=0$
- $n! = n \times (n-1)!$ if $n>0$

A factorial can be computed:

- Recursively, by calling the function within itself.
- Non-Recursive, using a for loop

→ Algorithm : ① Start the program.

② Define a recursive function fact_recursive(num)

- if num == 0, return 1

code :

```
#include <stdio.h>
int fact_recursive(int num)
{
  if (num==0)
      return 1;
  else
      return num* fact_recursive(num-1);
}
int fact_nonrecursive(int num)
{
   int f=1, i;
   for(i=1; i<=num; i++)
   f=f*i;
  return f;
}
void main()
{
   int n,r;
   int nCr_rec, nCr_nonrec
   printf("Enter values of n and r: ");
   scanf("%d %d", &n, &r);
   if(r>n)
   { printf("Invalid input!");
   nCr_rec = (fact_recursive(n) / (fact_nonrecursive(r) *
                                   fact_nonrecursive(n-r));

   printf("using recursive function:");
   printf("%d %d = %d\n", n, r, nCr_rec);
   printf("using non recursive function:");
   printf("%d %d = %d\n", n, r, nCr_nonrec);
   }
```

- else return num* fact_recursive (num - 1)

③ Define a non recursive function fact_nonrecursive (num)
- initialize f=1
- use a loop to multiply numbers from 1 to num.
- return f

④ In main function :
- read values of n and r.
- compute factorial using both methods
- compute binomial coefficient
- display the results with suitable message.

⑤ End the program

→ output

Enter values of n and r : 6 , 5

Using Recursive function :
C(6,5) = 6

Using Non-Recursive function :
C(6,5) = 6

CLASSMATE
Date :
Page :

② Develop a recursive function GCD (num1, num2) that accepts 2 integer arguments. Write a C program that invokes this function to find the greatest common divisor of 2 given integers.

→ Aim : To write a C program to find the GCD of two numbers using a recursive function.

→ Theory : The GCD (greatest common divisor) of 2 integers is the largest positive integer that divides both numbers without leaving a remainder.

The recursive function continues until the second argument (b) becomes zero.

→ Algorithm :

① Start the program.

② Define a recursive function gcd(int a, int b)
if b == 0, return a,
else return gcd(b, a%b)

③ In main():
- declare and initialize 2 integers a and b.
- Call the function gcd(a, b) and store the result in a variable c.
- Now print the outcome of the given program and get the result.

④ End the program.

→Code :

```
#include <stdio.h>
int gcd (int, int);
int main
{
 int a = 12;
 int b = 18;
  int c = gcd (b,a);
  printf ("%d", c);
  }
  int gcd (int a, int b)
  {
   if (b == 0)
   return a;
   else
   return gcd (b, a % b);
  }
```

→output : 6

CLASSMATE
Date :
Page :

③ Develop a recursive function FIBO (num) that accepts an integer argument. Write a C program that invokes this function to generate the Fibonacci sequence up to num.

→ Aim : To write a C program that uses a recursive function FIBO(num) to generate the Fibonacci sequence up to a given number of terms.

→ Theory : The Fibonacci sequence is a series of numbers where each number is the sum of 2 preceding ones.
It starts as :
0, 1, 1, 2, 3, 5, 8, 13...
A recursive function calls itself repeatedly until a base condition is met (in this case, n==0 or n==1).

→ Algorithm : ① Start the program.
② Define a recursive function fibo (num)
- return 0 if num == 0
- return 1 if num == 1
- otherwise, returns fibo (num-1) + fibo (num-2)

③ In the main () function :
- Accept an integer n from the user.
- Use a loop from i=0 to i<n and print each term by calling fibo(i)

④ End the program.

→ code:

```
#include <stdio.h>
int fibo (int num)
{
    if (num == 0)
        return 0;
    else if (num == 1)
        return 1;
    else
        return fibo (num-1) + fibo (num-2);
}
void main ()
{
    int n, i
    printf (" Enter number of terms");
    scanf (" %d", &n);

    printf ("Fiboracci sequence up to %d terms", n );
    for (i=0; i<n; i++)
    {
        printf ("%d" ; fibo(i));
    }
}
```

→ output:

Enter no of terms: 8

Fiboracci sequence up to 8 terms:

0 1 1 2 3 5 8 13

④ Develop a C function ISPRIME (num) that accepts an integer argument and return 1 if the argument is prime, a 0 otherwise. Write a C program that invokes this function to generate prime number b/w the given ranges.

→ Aim : To write a C program that defines a function ISPRIME (num) to check whether a number is prime or not.

→ Theory : A prime number is a natural number greater than 1 that has exactly two factors - 1 and itself.

For example : 2, 3, 5, 7, 11, 13...

To check if a number (n) is prime:
- if $n <= 1$, it is not prime
- For $n > 1$, check if any number b/w 2 and n/2 divides n.
  - if yes → then prime
  - if no → prime

The function ISPRIME (num) returns:
- 1 if the number is prime
- 0 if the number is not prime

→ Algorithm :
① Start the program
② Define function ISPRIME (int num)
- if num <= 1, return 0

→ code :

```
#include <stdio.h>
int ISPRIME (int num)
{
  int i;
  if (num <=1)
    return 0;
  for (i= 2; i<= num /2 ; i++)
  {
     if (num %i == 0)
      return 0;
  }
   return 1;
}
  void main ()
{
   int low , high , i ;
   printf ("Enter lower limit : ");
   scanf ("%d", &low);
   printf ("Enter higher limits : ");
    scanf ("%d", &high );

   printf (" prime number b/w %d and %d are
            : \n", low , high );
   {
      if (ISPRIME (i))
           printf ("%d", i);
      }
}
```

- loop from i = 2 to num/2.
  if num % i == 0, return 0.
- Return 1 (if no divisor found).

③ In the main() function:
- Accept two integer low and high (range limits).
- For each number from low to high call ISPRIME
- if the num / function returns 1, print the number

④ End the program

→ Result: The program successfully identifies and displays all prime numbers b/w the given range using a user defined function ISPRIME(num)

→ Output:

Enter the lower limit: 10
Enter the upper limit: 30

Prime numbers b/w 10 and 30 are:
11, 13, 17, 19, 23, 29

Expt. No.

Page No.
Date.

Experiment-7

Q) Write a C program that uses functions to perform the following operations.

① Reading a complex number.
② Writing a complex number.
③ Addition and Substraction of 2 complex numbers.

→ Aim: To read, print, add and substract complex numbers using structures.

→ Theory: A structure is / can group real and imaginary parts of a complex number. Functions perform operations.

→ Algorithm: ① Start program
② Read two complex numbers
③ Add them
④ get the output
⑤ Substract them
⑥ get the output
⑦ Display result

→ Result: Complex number addition and substraction done succesfully

Teacher's Signature .....

code:

```
#include <stdio.h>
struct complex {
  float real;
  float complex;
};
struct Complex read complex() {
  Struct complex c;
  printf("Enter real part: ");
  scanf("%f", &c.real);
  printf("Enter imaginary part:");
  scanf("%f", &c.imag);
  return c;
}
void printComplex (struct Complex c) {
  printf("%.2f + %.2fi\n", c.real, c.imag);
}
struct complex add (struct complex a, struct complex b) {
    struct Complex result;
    result.real = a.real + b.real;
    result.imag = a.imag + b.imag;
    return result;
}
struct complex subtract (struct complex a, struct complex b) {
  struct complex results;
  result.real = a.real - b.real;
  result.imag = a.imag - b.imag;
  return result;
}
int main() {
  struct complex c1, c2, sum, diff;
  printf("Enter first complex number: \n");
    c1 = read complex();
  printf("Enter second complex number: \n");
    c2 = read complex();
    sum = add(c1, c2);
    diff = substract(c1, c2);
    printf("Sum: ");
    printfComplex(sum);
  printf("Difference: ");
    printf complex(diff);
  return 0;
}
```

output :

```
Enter 1st imaginary no:
Enter real part : 3
Enter imaginary part: 4
Enter 2nd imaginary no: 2
Enter real part 2
Enter imaginary part : 5

Sum : 5.00 + 9.00i

Difference 1.00 + (-1.00i)
```

Expt. No.

Page No.
Date.

Q) Write a C program to compute the monthly pay of 100 employees using each employee's name, basic pay. The DA computed as 52·1· of the basic pay. Gross salary (basic pay + DA). Print the employees name and gross salary.

→ Aim: To calculate gross salary of 100 employees using structures.

→ Theory: Structures stores employees data
DA = 52·1· of basic pay
Gross = basic + DA

→ Algorithm:
① Start the program
② Enter the employee's name and basic pay.
③ Read employee name and basic pay
④ Compute DA and gross salary
⑤ Display name and gross salary
⑥ End the program

→ Result: Employee gross salaries computed and displayed.

OXFORD

Teacher's Signature .....

Code :

```
#include <stdio.h>
struct Employee {
  char name[30];
  float basic, da, gross;
};
int main() {
 struct Employee emp[100];
  int i;
 for(i=0; i<100; i++) {
 printf("Enter name of employee %d: ", i+1);
 scanf("%s", emp[i].name);
 printf("Enter basic pay: ");
 scanf("%f", &emp[i].basic);
 emp[i].da = 0.52 * emp[i].basic;
 emp[i].gross = emp[i].basic + emp[i].da;
 }
 printf("\nEmployee Name\tGross salary\n");
   for(i=0; i<100; i++) {
    printf("%s\t\t%.2f\n", emp[i].name, emp[i].gross);
  }
  return 0;
}
```

output

```
Enter name of employee 1 : Anjali
Enter basic pay : 10000
```

Employee name	Gross salary
Anjali	15200.00

Expt. No.

Page No.
Date.

3

Q) Create a book structure as a function argument, containing book_id, title, author name and price. Write a C program to pass the structure and print the book details.

→ Aim: To pass a structure to a function and print the book details.

→ Theory: A structure variable can be passed to a function by value.

→ Algorithm:
1. Start the program
2. Read book details
3. Pass structure to function
4. Display details
5. End the program

→ Result: Book details displayed using function

→ output:

Enter book ID: 101
Enter Title: C_ programming
Enter Author Dennis
Enter Price: 350

Teacher's Signature .....

Code:

```
#include <stdio.h>
struct Book {
  int book_id;
  char title[50];
  char author[30];
   float price;
};
void display(struct Book b) {
  printf("\nBook ID: %d\n", b.book_id);
  printf("Title: %s\n", b.title);
  printf("Author: %s\n", b.author);
  printf("Price: %.2f\n", b.price);
}
int main() {
    struct Book b;
    printf("Enter Book ID: ");
    scanf("%d", &b.book_id);
    printf("Enta Title: ");
    scanf("%s", b.title);
    printf("Enter Author: ");
    scanf("%s", b.author);
     printf("Enter Price: ");
     scanf("%f", &b.price);
  display(b);
  return 0;
}
```

Expt. No.

Page No.
Date.

4

Q) Create a union containing 6 strings : name, home_address, hostel_address, city, state and zip. Write a C program to display your address

→ Aim : To store and display present address using union.

→ Theory : Union shares one memory location for all members; only one active at a time

→ Algorithm :
① Start the program.
② Store present address in union
③ Display the address
④ End the program

→ Result : Present address displayed successfully.

OXFORD

Teacher's Signature .....

Code:

```
#include <stdio.h>
union Address {
    char name[30];
    char home_address[50];
    char hostel_address[50];
    char city[20];
    char state[20];
    char zip[10];
};
int main() {
    union Address addr;
    printf("Enter present address: ");
    scanf("%s", addr.hostel_address);
    printf("Present address: %s", addr.hostel_address);
    return 0;
}
```

output:

Enter present Ho address : Hostel Tulips

Present address : Hostel Tulips

CLASSMATE
Date :
Page :

Experiment-8
Pointers

① Declare different types of pointers (int, float, char) and initialize them with the address of variables. Print the values of both the pointers and the variables they point to.

→ **Aim**: To declare different types of pointers (int, float, char) and initialize them with the address of variables. Then, print the values of both the pointers

→ **Theory**: A pointer is a variable that stores the memory address of another variables. Pointers are used for dynamic memory allocation, function arguments by reference and accessing data efficiently.
- datatype specifies the type of data the pointer will point to.

→ **Algorithm**:
① Start the program.
② Declare 3 variables of types int, float, char
③ Declare corresponding pointers of types int*, float*, char*
④ Assign the addresses of the variables of their respective pointer's using the address of operator &.
⑤ Print
- The addresses stored in the pointers.
- The values of the variables directly
- The values of the variables using pointers (dereferencing)

→ code :

```
#include <stdio.h>
void main()
{
    int a = 10;
    float b = 5.25;
    char c = 'A';

    int *ptr1 = &a;
    float *ptr2 = &b;
    float *ptr3 = &c;

    printf("Values of variables : \n");
    printf("a = %d \n", a);
    printf("b = %.2f \n", b);
    printf("c = %c \n\n", c);

    printf("Addresses stored in pointers : \n");
    printf("ptr 1 = %p \n", ptr1);
    printf("ptr 2 = %p \n", ptr2);
    printf("ptr 3 = %p \n\n", ptr3);

    printf("Values accessed using pointers : \n");
    printf("*ptr 1 = %d \n", *ptr1);
    printf("*ptr 2 = %.2f \n", *ptr2);
    printf("*ptr 3 = %c \n", *ptr3);
}
```

→ **Result:** The program successfully demonstrates the declaration, initialization and usage of different types of pointers (int, float, char).

→ **Output:**

Values of variables:

a = 10
b = 5.25
c = A

Addresses of / stored in pointers:

ptr 1 = 0x7ggf68421a4
ptr 2 = 0x7ggf68421a8
ptr 3 = 0x7ggf68421ab

Values accessed using pointers:

*ptr 1 = 10
*ptr 2 = 5.25
*ptr 3 = A

CLASSMATE
Date :
Page :

② Perform pointer arithmetic (increment and decrement) on pointers of different data types. Observe how the memory addresses change and the effects on data access.

→ Aim: To perform pointer arithmetic on pointers of different data types and observe how memory address change and how it affects data access.

→ Theory: Pointer arithmetic allows operations like increment, decrement, addition and substraction on pointers.
When a pointer is incremented or decremented the address it holds changes by an amount equal to the size of the data type it points to.

→ Algorithm: ① Start the program
② Declare variables of types int, float, char.
③ Declare pointers and assign them the address of the variable.
④ Display the initial addresses stored in the pointers.
⑤ Increment each pointer and display the new address.
⑥ Decrement each pointer and display the updated address again.
⑦ Observe how the addresses change according to data type size.
⑧ End the program

→ code :

```
#include <stdio.h>
void main()
{
  int a= 10;
  float b= 5.5;
  char c= 'x';

  int *p1 = &a;
  float *p2= &b;
  char *p3 = &c;

printf(" Initial pointer addresses : \n ");
printf(" p1 = %p \n" , p1);
printf (" p2 = %p \n" , p2);
printf (" p3= %p \n\n", p3);
```

(increment operators)

```
p1++
p2++
p3++

printf (" after incrementing pointers : \n ");
printf (" p1 = %p \n" , p1);
printf (" p2 = %p \n" , p2);
printf (" p3 = %p \n\n", p3);
```

(decrement operator)

```
p1--
p2--
p3--

printf (" After decrementing pointers : \n");
printf (" p1 = %p \n" , p1);
printf (" p2 =%p \n" , p2);
printf (" p3 =%p \n" , p3);
}
```

→ **Result:** The program successfully demonstrates pointer arithmetic, showing that incrementing or decrementing a pointer changes its address based on the size of the data type it points to

→ **output:**

Initial pointer addresses:

p1 = 0x7ffceb6e91a4
p2 = 0x7ffceb6e91a8
p3 = 0x7ffceb6e91aC

After incrementing pointers:

p1 = 0x7ffceb6e91a4 (increased by 4 bytes)

p2 = 0x7ffceb6e91ac (increased by 4 bytes)

p3 = 0x7ffceb6e91ad (increased by 1 byte)

After decrementing pointers:

p1 = 0x7ffceb6e91a4
p2 = 0x7ffceb6e91a8
p3 = 0x7ffceb6e91a6

CLASSMATE
Date :
Page :

③ Write a function that accepts pointers as parameters. Pass variables by reference using pointers and modify their values within the function.

→ Aim : To write a function that accepts pointers as parameters and modifies the values of variables passed by reference using pointers.

→ Theory : Call by value → a copy of the variable is passed. changes don't affect the original variable
Call by reference → the address of the variable is passed using pointers, allowing the function to directly modify the original variable.

→ Algorithm : ① Start the program
② Declare 2 integers a and b
③ Define a function
④ In main() print the original values of a and b
⑤ Call the function using its address
⑥ Print the updated values after modification
⑦ Stop the program.

→ Result : The program successfully demonstrates passing variable by reference using pointers and shows how their values can be modified within a function.

→ code :

```
#include<stdio.h>
void modify values (int *x , int*y)
{
    *x = *x +10;
    * y= *y *2 ;

}
void main ()
{
   int a= 5,  b=7;
  printf (" Before modification : \n");
  printf (" a= %d \n =%d \n\n", a, b);

  modify values (&a ,&b);

  printf (" after modification : \n");
   printf (" a =%d \n = %d \n", a, b);

}
```

→ output

Before modification
a = 5
b = 7

After modification
a = 15
b = 14

Anjali Kalia
590024902
B-20

CLASSMATE
Date :
Page :

Experiment - 9

File Handling in C

① Write a program to create a new file and write a text into it.

→ Aim : To write a C program that creates a new file and stores user-entered text into it.

→ Theory : File handling in C helps store data permanently on the disk. To work with files, C uses a special pointer of type FILE*
The following functions are used:

- fopen → opens / creates new file
- fputs() → writes a string to a file
- fclose() → closes the file after use
- File mode "w" → creates a new file for writing (overwrites if it exists)

→ Algorithm :
① Start the program
② Declare the file pointer of type FILE*
③ Open the file in write mode ("w") using fopen()
④ Check if the file is successfully created.
⑤ Accept text input from the user.
⑥ Write the text to the file using fputs().
⑦ Close the file using fclose().
⑧ End the program.

Code:

```
#include<stdio.h>
int main()
{
  FILE*
 char a[100];
f = fopen("abc.txt", "w");
if (f == NULL)
{
  printf("nothing");
  return 0;
 }
 fgets (a, sizeof (a), stdin);
 fputs (a, f);
 fclose (f);
 return 0;
}
```

Output

```
Enter text:
Hello World
```

[The text entered by the user is succesfully written into the file "abc.txt")

→ Result : The program successfully created a new file named abc.txt and stored the user entered text "Hello World" into it. This demonstrates the basic file handling operations.

CLASSMATE
Date :
Page :

② Open an existing file and read its content character by character and then close the file.

→ Aim: To open the C program file and read its contents character by character using fgetc()

→ Theory: Character-level file reading in C is performed using the function:

- fgetc(FILE *fp)
  reads a single character from the file and returns it. when end of the file reached, it returns EOF.

→ Algorithm:
① Start the program
② Declare a file pointer FILE *f
③ Open the file in read mode using fopen()
④ If the file fails to open, display an error.
⑤ Use a loop to read characters using fgetc(f)
⑥ Print each character to the screen.
⑦ Continue until EOF is reached.
⑧ Close the file using f(close()).
⑨ Stop the program.

→ Result: The program successfully opened the file and displayed its contents character by character using fgetc(). This demonstrates how file can be read at the lowest level in C.

code :

```
#include <stdio.h>
{
FILE *f;
char a;
f = fopen("abc.txt", "r");
if (f = NULL)
{
  printf("nothing");
  return 0;
}
while ((a=fgetc(f))! = EOF)
{
printf("%c", a);
}
  fclose(f);
}
```

output :

```
Enter text :
    Hello World
```

(Now, this content of the file saved in abc.txt is displayed character by character on the screen.)

" H e l l o W o r l d "

CLASSMATE
Date :
Page :

③ Open a file, read its content line by line and display each line on the console.

→ Aim: To write a C program that opens an existing file and reads its contents line by line using fgets()

→ Theory: Line based reading is one of the most common file handling operation in C.
The function used is :
fgets (char *str, int size, FILE* fp)
read one full line from the file into a string.
fgets :
i) Reads an entire line at once.
ii) Prevents buffer overflow.
iii) Easy to process text file.

→ Algorithm : ① Start the program.
② Declare a file pointer FILE *f and character array.
③ open the file in read mode using fopen().
④ If the file cannot be opened, display an error.
⑤ Use a loop with fgets() and read each line
⑥ Print each line on the screen.
⑦ Continue until the end of file is reached.
⑧ Close the file.
⑨ Stop the program.

→ Result : The program successfully opened the file and displayed its contents line by line using fgets(). This demonstrates effective line-based reading in C.

code :

```
#include<stdio.h>
int main()
{
  FILE *f;
  char a[100];
  f = fopen("abc.txt", "r");
  if (f== NULL)
  {
    printf("nothing");
    return 0;
  }
  while (fgets(a, sizeof(a), f) != NULL)
  {
    printf("%s", a);
  }
  fclose(f);
  return 0;
  }
```

output :

```
Enter a text : Hello World
Enter a text : This is a sample code
```

```
Hello World
This is a sample code
```

(The content of the file is displayed line to line)

CLASSMATE
Date :
Page :

Experiment-10

Dynamic Memory Allocation

① → Write a program to create a simple linked list in C using pointer and structure.

→ Aim : To create a simple linked list in C using pointer and structure.

→ Theory : A linked list is a dynamic data structure consisting of nodes.
Each node contains :
1) Data
2) Pointer to next node

Nodes are created at runtime using malloc(), which allocates memory.

→ Algorithm :
① Start the program
② Define a struct node with data and next pointer.
③ Use malloc() to allocate memory for each node.
④ Store data inside each node.
⑤ Link nodes together using pointers.
⑥ Traverse the list and display all nodes
⑦ End the program

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* next;
};
int main() {
  struct Node * head, *second, * third
  head = (struct Node*) malloc (sizeof (struct Node));
  second = (struct Node*) malloc (sizeof (struct Node));
  third = (struct Node*) malloc (sizeof (struct (Node));
  head - data = 10;
  head - next = second;
  second - data = 20;
  second - next = third;
  third - data = 30;
  third - next = NULL;
  struct Node* temp = head
  while (temp != NULL) {
    printf ("%d", temp - data);
    temp = temp - next;
  }
  printf ("NULL");
  return 0;
}
```

→ Result : A simple singly linked list was successfully created using structures, pointers and dynamic memory allocation.

→ output : 10 - 20 - 30 - NULL

CLASSMATE
Date :
Page :

② Write a program to insert item in middle of the linked list

→ Aim: To write a C program that inserts a new node in the middle of a singly linked list.

→ Theory: To insert in the middle:
① Traverse the list to the node just before insertion point.
② Create a new node using malloc()
③ Adjust the pointers

→ Algorithm: ① Start the program.
② Create a linked list with a few nodes.
③ Take the position where a new data is to be inserted.
④ Traverse to the 1st position node.
⑤ Create a new node.
⑥ Update pointers to insert the new node.
⑦ Display the updated list
⑧ Stop the program

→ Result: A new node was successfully inserted in the middle of the linked list using dynamic memory allocation.

→ output: Before: 10 - 20 - 30 - NULL
After: 10 - 15 - 20 - 30 - NULL

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data
  struct Node* next;
};
void intersectionMiddle (struct Node * head, int value) {
  struct Node * newNode = (struct Node*) malloc (sizeof (struct Node));
   newnode — data = value;
   struct Node * temp = head;
   int count = 0;
   while (temp != NULL) {
    temp = temp - next;
      count++
  }
   int mid = count/3;
    temp = head;
   for (int i = 1; i < mid ; i++) {
      temp = temp-next
    }
   New node -next = temp -next
   temp -next = new Node;
}
void display (struct Node* head) {
     struct Node* temp = head;
     while (temp != NULL) {
       printf ("%d - ", temp_data);
        temp = temp - next;
      }
      print f ("NULL);
   }
   int main () {
   Struct Node * head, * second, * third
   printf ("Before insertion \n");
    display head);
} return 0;
```

Experiment-12

Preprocessor and Directives in C

① Write a program to define some constant variable in preprocessor.

→ Aim : To write a program to define some constant variable in preprocessor.

→ Theory : The C preprocessor source code before compilation.

The directive :

① Define symbolic constants
② Replaces all occurrences of the symbol with its value.
③ Has no datatype
④ Helps improve readability and maintainability.

```
#define PI 3.14
```

This makes PI act like a constant throughout the program.

→ Algorithm : ① Start the program
② Use #define to declare a constant.
③ Use the constant in a calculation
④ Display the result.
⑤ Stop the program.

Code:

```
#include <stdio.h>
#define PI 3.14
int main() {
    float r = 5;
    float area = PI * r * r ;
    printf (" Area = %.2f", area);
    return 0;
}
```

output:

Area : 78.50

→ Result : The program successfully demonstrates the use of #define to create a symbolic constant using the C preprocessor.

CLASSMATE
Date :
Page :

② Write a program to define a function in directives.

→ Aim : To write a C program that defines a function-like operation using a preprocessor macro.

→ Theory : Preprocessor macros can mimic functions.

```
#define SQUARE(x) (x * x)
```

- No type checking
- Replaced directly by text
- Faster than normal functions

→ Algorithm :
1. Start the program
2. Define a function-like macro using #define
3. Accept or initialize a number.
4. Pass the number to the macro.
5. Display the result
6. End the program.

→ Result : The program successfully implemented a function-like macro using the preprocessor directive #define

code :

```
#include <stdio.h>
# define SQUARE(x) ((x) * (x))
  int main() {
      int num = 6;
  printf("Square of %d is: %d\n", num, SQUARE(num));
  return 0;
}
```

output :

Square of 6 : 36

CLASSMATE
Date :
Page :

EXPERIMENT-13

Macros in C

① Write a program to define multiple macro to perform arithemetic functions.

→ Aim : To write a C program that defines multiple macros using the preprocessor and uses them to perform arthematic operations.

→ Theory : A macro in C is defined using the preprocessor directives
#define
Macros are of 2 types :
① Object like macros - simple constant
② Function like macros - behave like inline function

→ Algorithm : ① Start the program
② Define macros for arithematic operations using #define
③ Declare 2 numbers
④ Use macros to compute results
⑤ Display the results.
⑥ End the program

→ Result : The program successfully used multiple functions like macros to perform arithemtic operations.

Code :

```
#include <stdio.h>
# define  ADD (a,b) ((a) + (b))
# define  SUB (a,b) ((a) - (b))
# define  MUL (a,b) ((a) * (b))
# define  DIV (a,b) ((a) / (b))

int main() {
int x= 20, y= 5;
printf("Addition = %d\n", ADD(x,y));
printf("substraction = %d\n", SUB(x,y));
printf("Multiplication= %d\n", MUL(x,y));
printf("Multiplication = %d\n", DIV(x,y));

return 0;
}
```

output :

Addition : 25

Substraction : 15

Multiplication : 100

Division : 4

Experiment-14

Static Library in C

① Write a program to create static library for performing arithmetic functions / any other program.

→ **Aim:** 1) To create a static library in C containing arithmetic functions.
2) To write a program that uses static library in another C program.

→ **Theory:** A static library is a collection of object file bundled together into a single file with the extention .a (archive file)

Steps to create a static library:

① Write the function declaration in a header file (arith.h)
② Write function definitions in source file (arith.c)
③ Compile to object file using,
gcc -c arith.c
④ Create library using,
ar rcs libarith.a arith.o
⑤ Link the library with the main program.

→ **Algorithm:** ① Write arithmetic functions (sum, sub) in arith.c
② Write function prototypes in arith.h
③ Compile the file into object code using gcc c
④ Use ar rcs to create the static library libarith.a

Code :

```
#include <stdio.h>
#include arith.h
int main()
{
int a = 20;
int b= 20;
printf("%d ", sum(a,b))
printf(" %d", sub(a,b))
}
main.c
#include "arith.h
int sum (int a, int b)
{
return a+b;
}
int sub( int a,int b)
{
return a+b;
}
int sub (int a,int b)
{
return a -b;
}
arith.c
# ifndf
# define ARITH
int sum ( int a, int b);
int sub (int a, int b);
# endif
```

Output :

```
50
-10
```

CLASSMATE
Date :
Page :

→ **Result** : The static library libarith.a was successfully created and used in another program. Arithmetic function (sum & sub) were accessed by linking the static library with the main C program

Experiment-15

Shared library in C

① Write a program to create a shared library for performing arithmetic function / other program.

→ **Aim** : 1) To create a shared library in C for performing arithmetic function
2) To write a program that uses this shared library.

→ **Theory** : A shared library is a file containing reusable code that is linked at runtime instead of compile-time.

- File extension is .so (shared object)
- Loaded into memory when program runs.
- Saves memory because one copy is shared by multiple programs.
- Reduces executable size
- Suitable for modular and scalar applications

Commands used:

1) gcc -fPIC -c file.c - needed for shared libraries
2) gcc -shared -o libname.so file.o - create shared library
3) gcc main.c -L. -lname -o output - liked shared library
4) export LD_LIBRARY_PATH=$LD_LIBRARY_PATH - make library available for runtime

→ **Algorithm** : ① Write arithmetic functions (sub, sum) in arith.c
② Write functions prototypes in arith.h

(A) Header file

```
#ifndef ARITH
#define ARITH

    int sum(int a, int b);
    int sub(int a, int b);

#endif
```

(B) Library Source file

```
#include "arith.h"

int sum (int a, int b) {
  return a+b;
}
int sub (int a, int b) {
  return a-b;
}
```

(C) Main program

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

int main() {
 int a = 20;
 int b = 30;
 printf("%d\n", sum(a,b));
 printf("%d\n", sub(a,b));
 return 0;
}
```

-> Output

50

-10

③ Compile source file using -fPIC for position-independent code.
④ Create shared library libarith.so using -shared.
⑤ Set library path or export environment variable.

* To run the program : ./output

→ Result : A shared library (libarith.so) was successfully created and used in another C program. The program dynamically linked the arithmetic functions at runtime and executed correctly.