C LAB REPORT

Experiment1 – Installation, Environment Setup & Basic C Programs

Q1. Write a C program to print “Hello World”.

Aim

To write a C program that displays **Hello World** on the screen.

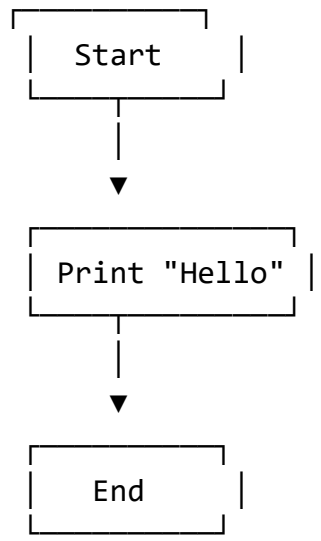
Algorithm

1. Start
2. Include stdio.h
3. Use `printf()` to print the message
4. End

Pseudocode

```
BEGIN
  PRINT "Hello World"
END
```

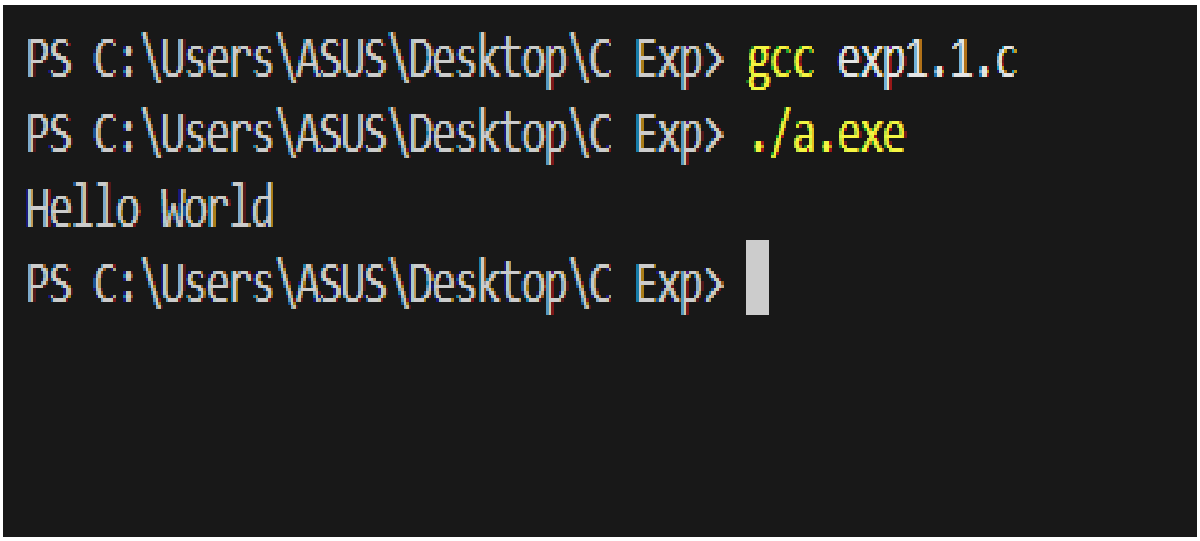
Flowchart (Text Format)



C Program

```c
#include <stdio.h>

int main() {
    printf("Hello World");
    return 0;
}
```

Output



Conclusion

The program successfully prints the message “Hello World”.

Q2. Write a C program to print the address in multiple lines using newline characters.

Aim

To print an address using \n for new lines.

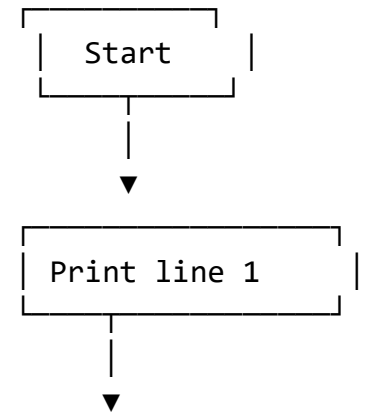
Algorithm

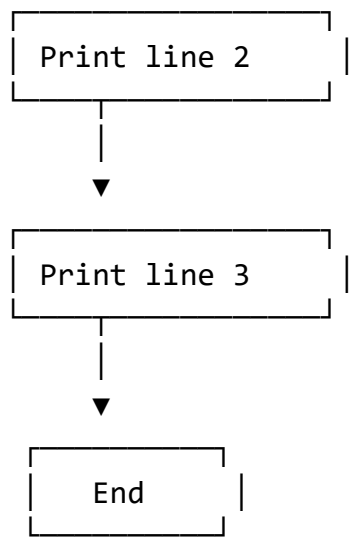
1. Start
2. Use `printf()` with newline characters
3. Print each line of the address
4. End

Pseudocode

```
BEGIN
  PRINT "House No. 123"
  PRINT "Sector 10"
  PRINT "New Delhi"
END
```

Flowchart





C Program

```
#include <stdio.h>

int main() {
    printf("House No. 123\n");
    printf("Sector 10\n");
    printf("New Delhi\n");
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp1.2.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Omkareshwar Chaubey
Inarbharwa
PS:-Ramnagar, PO:-Jogia
West Champaran
Bihar
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

The program successfully prints the address on multiple lines.

Q3. Write a C program that prompts the user to enter their name and age.

Aim

To read and display the user's name and age using input functions.

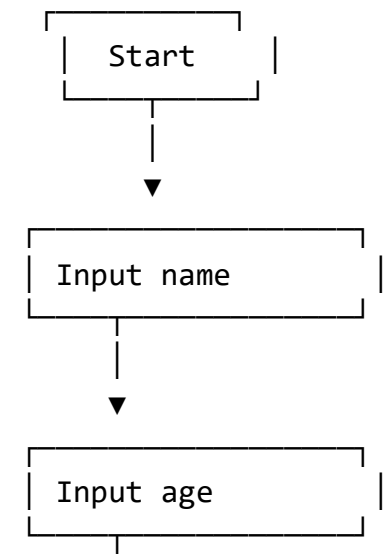
Algorithm

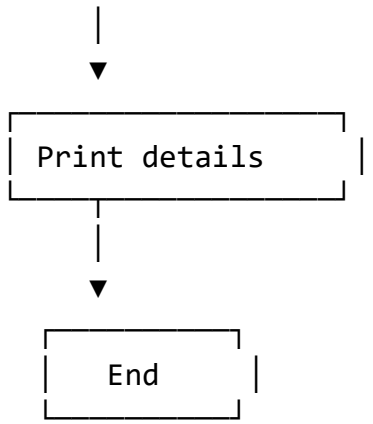
1. Start
2. Declare variables: name, age
3. Ask the user to enter name
4. Ask the user to enter age
5. Display the entered details
6. End

Pseudocode

```
BEGIN
  DECLARE name, age
  INPUT name
  INPUT age
  PRINT name and age
END
```

Flowchart





C Program

```
#include <stdio.h>

int main() {
    char name[20];
    int age;

    printf("Enter your name: ");
    scanf("%s", name);

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("Your name is %s and your age is %d\n", name, age);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp1.3.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter your name: Omkareshwar
Enter your age: 17
Hello Omkareshwar!
You are 17 years old.
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

The program accepts user input and prints it correctly.

Q4. Write a C program to add two numbers taken from the user.

Aim

To input two numbers and display their sum.

Algorithm

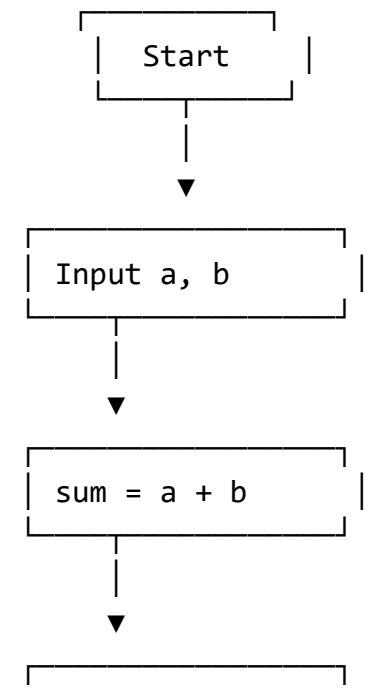
1. Start

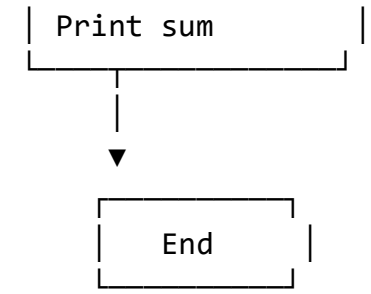
2. Declare variables a, b, sum
3. Ask user for two numbers
4. Read input
5. sum = a + b
6. Display sum
7. End

Pseudocode

```
BEGIN
  INPUT a
  INPUT b
  sum = a + b
  PRINT sum
END
```

Flowchart





C Program

```
#include <stdio.h>

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

    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    sum = a + b;

    printf("Sum = %d\n", sum);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp1.4.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter first number: 4
Enter second number: 5
The sum of 4 and 5 is 9
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

The program successfully calculates and displays the sum of two numbers.

EXPERIMENT2: OPERATORS

Q1. Write 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 using user-input length and width.

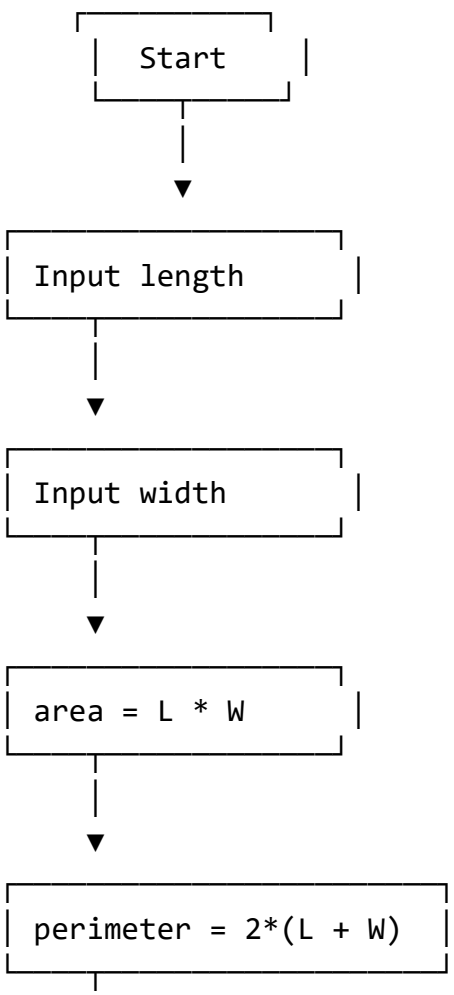
Algorithm

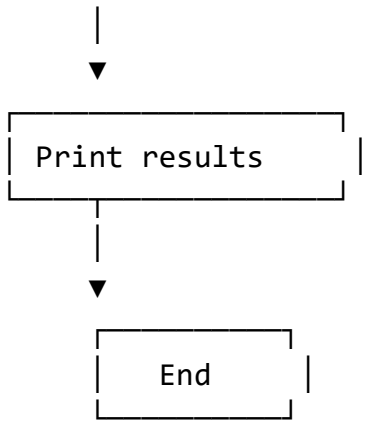
1. Start
2. Declare variables: length, width, area, perimeter
3. Ask the user for length
4. Ask the user for width
5. Compute area = length × width
6. Compute perimeter = 2 × (length + width)
7. Display area and perimeter
8. End

Pseudocode

```
BEGIN
  INPUT length
  INPUT width
  area = length * width
  perimeter = 2 * (length + width)
  PRINT area
  PRINT perimeter
END
```

Flowchart





C Program

```
#include <stdio.h>

int main() {
    float length, width, area, perimeter;

    printf("Enter length of rectangle: ");
    scanf("%f", &length);

    printf("Enter width of rectangle: ");
    scanf("%f", &width);

    area = length * width;
    perimeter = 2 * (length + width);

    printf("Area = %.2f\n", area);
    printf("Perimeter = %.2f\n", perimeter);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp2.1.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter length of the rectangle: 4
Enter width of the rectangle: 5

Area of Rectangle = 20.00
Perimeter of Rectangle = 18.00
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

The program correctly calculates and displays the area and perimeter of a rectangle using arithmetic operators.

Q2. Write 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 arithmetic operators.

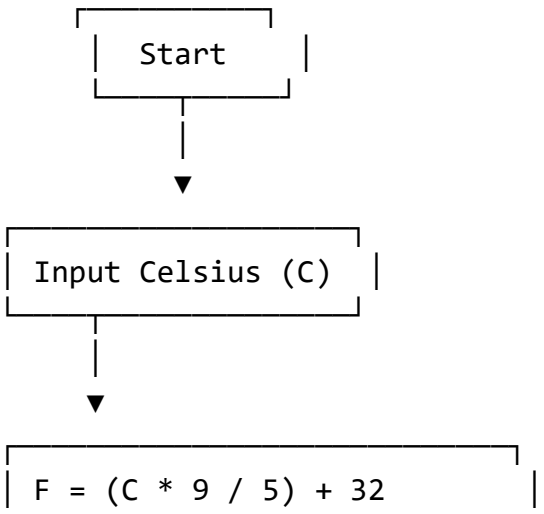
Algorithm

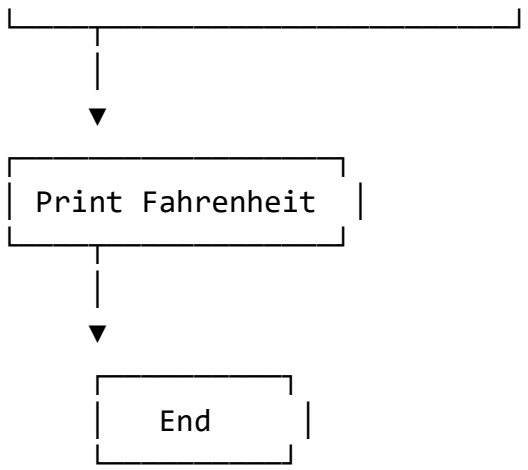
1. Start
2. Declare variables: Celsius, Fahrenheit
3. Input Celsius
4. Calculate Fahrenheit = (C × 9/5) + 32
5. Display Fahrenheit value
6. End

Pseudocode

```
BEGIN
  INPUT C
  F = (C * 9 / 5) + 32
  PRINT F
END
```

Flowchart





C Program

```
#include <stdio.h>

int main() {
    float celsius, fahrenheit;

    printf("Enter temperature in Celsius: ");
    scanf("%f", &celsius);

    fahrenheit = (celsius * 9 / 5) + 32;

    printf("Temperature in Fahrenheit = %.2f\n", fahrenheit);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp2.2.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter temperature in Celsius: 100
Temperature in Fahrenheit: 212.00
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

The program successfully converts temperature from Celsius to Fahrenheit using the appropriate arithmetic formula.

.

EXPERIMENT3.1: CONDITIONAL STATEMENTS

Q1. Write a program to check if the triangle is valid or not.

If valid, check whether it is Isosceles, Equilateral, Right-angled, or Scalene.

Take the sides as input from the user.*

Aim

To determine if a triangle with given sides is valid and classify its type.

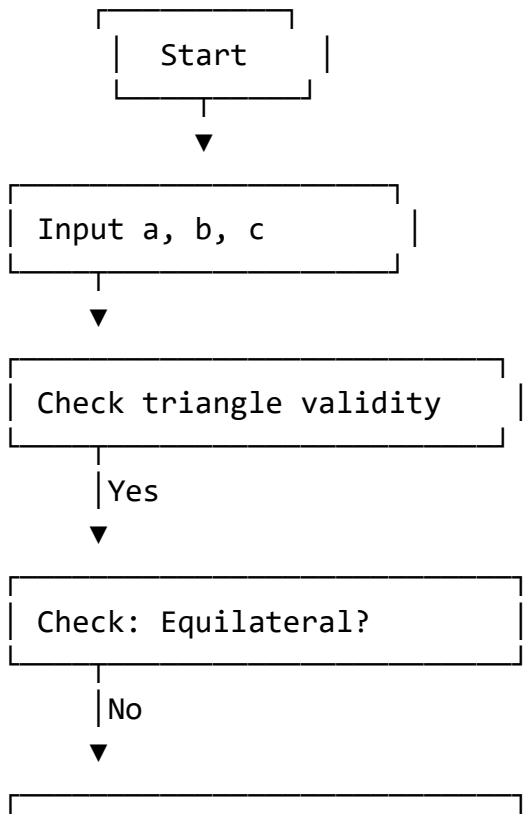
Algorithm

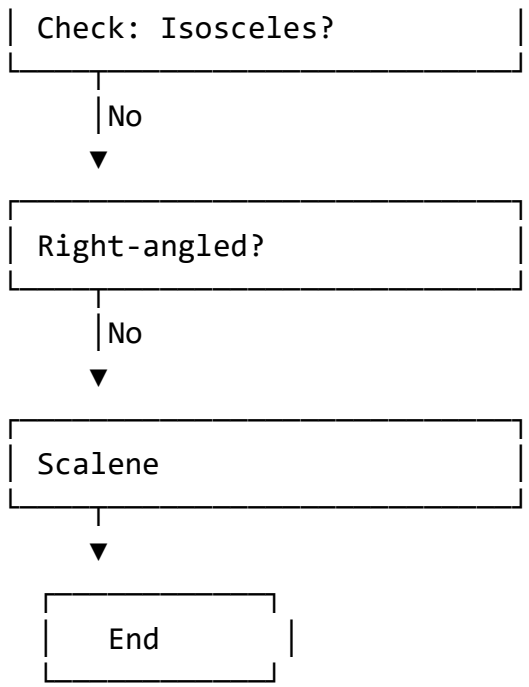
1. Start
2. Input sides a, b, c
3. Check **Triangle Validity Condition**:
    a. If (a + b > c) AND (b + c > a) AND (a + c > b)
4. If NOT valid → print “Invalid triangle”
5. Else:
    a. If all sides equal → Equilateral
    b. Else if two sides equal → Isosceles
    c. Else if ($a^2 + b^2 = c^2$) or similar → Right-angled
    d. Else → Scalene
6. End

Pseudocode

```
BEGIN
  INPUT a, b, c
  IF a + b > c AND b + c > a AND a + c > b THEN
        IF a = b AND b = c THEN PRINT "Equilateral"
        ELSE IF a = b OR b = c OR a = c THEN PRINT "Isosceles"
        ELSE IF (a*a + b*b = c*c) OR ... THEN PRINT "Right Angled"
        ELSE PRINT "Scalene"
  ELSE
        PRINT "Invalid Triangle"
END
```

Flowchart



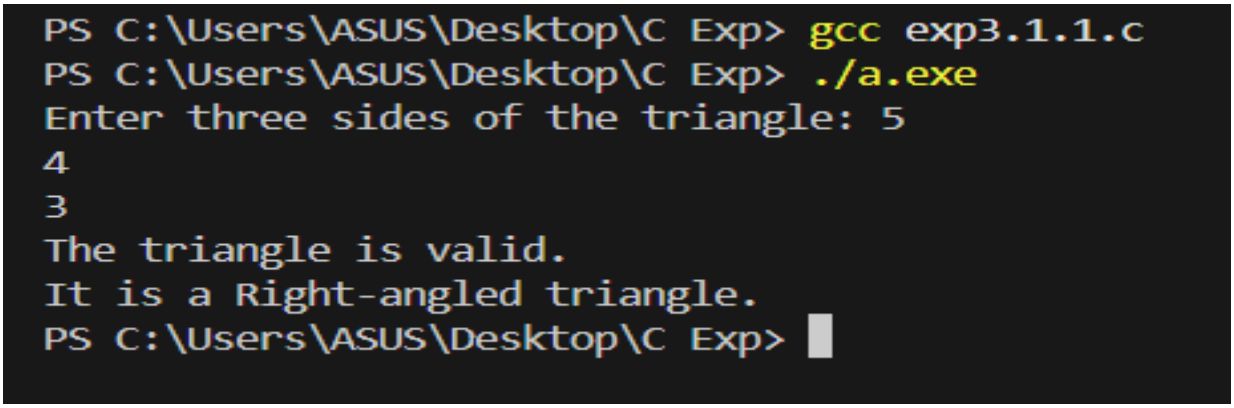


C Program

```
#include <stdio.h>

int main() {
    int a, b, c;
    printf("Enter three sides: ");
    scanf("%d %d %d", &a, &b, &c);

    if (a + b > c && b + c > a && a + c > b) {
        printf("Triangle is valid\n");

        if (a == b && b == c)
            printf("Equilateral Triangle");
        else if (a == b || b == c || a == c)
            printf("Isosceles Triangle");
        else if ((a*a + b*b == c*c) ||
                 (b*b + c*c == a*a) ||
                 (a*a + c*c == b*b))
```

```
            printf("Right Angled Triangle");
        else
            printf("Scalene Triangle");
    } else {
        printf("Invalid Triangle");
    }

    return 0;
}
```

Output



Conclusion

The program correctly identifies a valid triangle and classifies its type.

Q2. Write a program to compute BMI and print values according to categories.

BMI Formula:

[

BMI = \frac{Weight(kg)}{Height(m)^2}

]

Aim

To compute BMI and categorize it as per the given BMI ranges.

BMI Categories

BMI Range	Category
< 15	Starvation
15 – 17.5	Anorexic
17.5 – 18.5	Underweight
18.5 – 25	Ideal
25 – 30	Overweight
30 – 40	Obese
> 40	Morbidly Obese

Algorithm

1. Start
2. Input weight and height
3. Compute BMI

4. Check ranges using if-else ladder
5. Display the category
6. End

Pseudocode

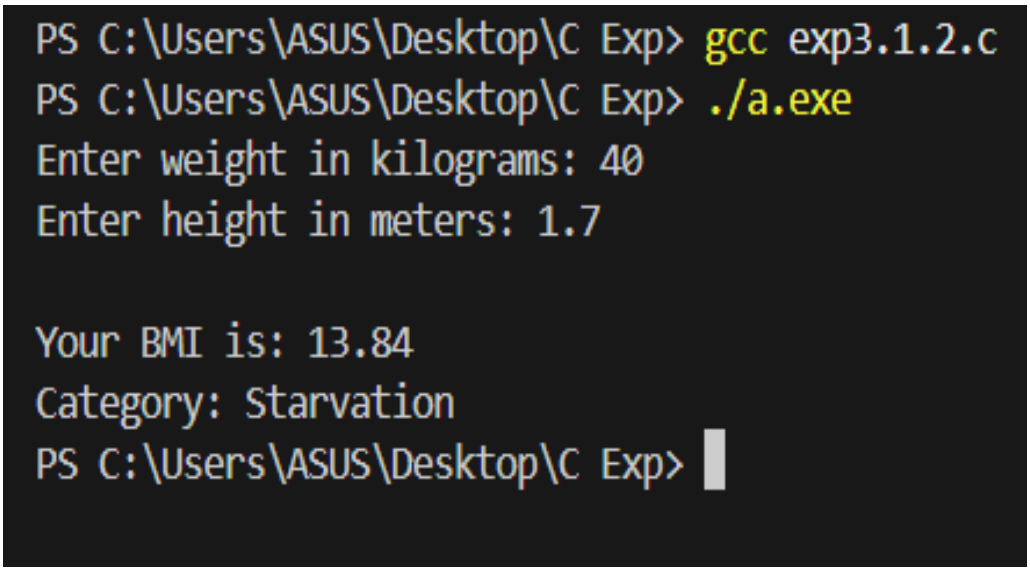
```
BEGIN
  INPUT weight, height
  BMI = weight/(height*height)
  IF BMI < 15 THEN Starvation
  ELSE IF BMI < 17.5 THEN Anorexic
  ELSE IF BMI < 18.5 THEN Underweight
  ELSE IF BMI < 25 THEN Ideal
  ELSE IF BMI < 30 THEN Overweight
  ELSE IF BMI < 40 THEN Obese
  ELSE Morbidly Obese
END
```

C Program

```
#include <stdio.h>

int main() {
    float weight, height, bmi;

    printf("Enter weight (kg): ");
    scanf("%f", &weight);

    printf("Enter height (m): ");
    scanf("%f", &height);

    bmi = weight / (height * height);

    printf("BMI = %.2f\n", bmi);
```

```
    if (bmi < 15)
        printf("Starvation");
    else if (bmi < 17.5)
        printf("Anorexic");
    else if (bmi < 18.5)
        printf("Underweight");
    else if (bmi < 25)
        printf("Ideal");
    else if (bmi < 30)
        printf("Overweight");
    else if (bmi < 40)
        printf("Obese");
    else
        printf("Morbidly Obese");

    return 0;
}
```

Output



Conclusion

BMI is calculated correctly and categorized according to the given medical ranges.

Q3. Write a program to check if three points (x1,y1), (x2,y2), (x3,y3) are collinear.

Aim

To check whether 3 points lie on the same straight line using the area method.

Method (Area of Triangle = 0)

Points are collinear if:

[

(x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)) = 0

]

Algorithm

1. Input coordinates of the three points
2. Apply the formula
3. If expression = 0 → Collinear
4. Else → Not Collinear

Pseudocode

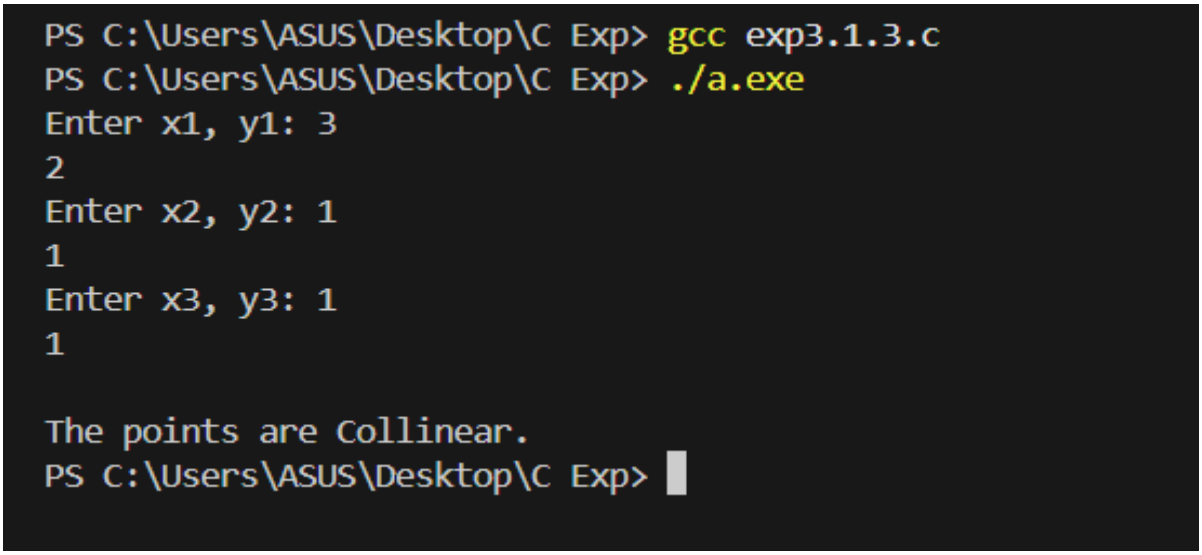
```
BEGIN
  INPUT x1,y1,x2,y2,x3,y3
  compute A = x1(y2–y3)+x2(y3–y1)+x3(y1–y2)
  IF A == 0 THEN collinear
  ELSE not collinear
END
```

C Program

```
#include <stdio.h>

int main() {
    int x1, y1, x2, y2, x3, y3;
    int area;

    printf("Enter three points (x y): ");
    scanf("%d %d %d %d %d %d", &x1,&y1,&x2,&y2,&x3,&y3);

    area = x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2);

    if (area == 0)
        printf("Points are collinear");
    else
        printf("Points are NOT collinear");
```

```
    return 0;
}
```

Output



Conclusion

The program correctly determines collinearity using the triangle area formula.

Q4. According to Gregorian calendar, 01-01-1900 was Monday.

If any year is input, find the day on 01-01 of that year.**

Aim

To determine the weekday of 1st January of any input year.

Logic

Count total days from 1900 to given year:

- Add 365 days per normal year
- Add 366 days per leap year
- Day index = total_days % 7

Mapping:

0 → Monday

1 → Tuesday

2 → Wednesday

3 → Thursday

4 → Friday

5 → Saturday

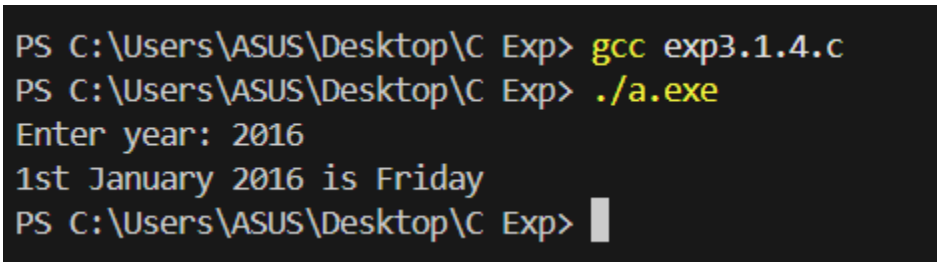
6 → Sunday

C Program

```
#include <stdio.h>

int main() {
    int year, y, days = 0;
```

```
    printf("Enter year: ");
    scanf("%d", &year);

    for (y = 1900; y < year; y++) {
        if ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0))
            days += 366;
        else
            days += 365;
    }

    int day = days % 7;

    switch(day) {
        case 0: printf("Monday"); break;
        case 1: printf("Tuesday"); break;
        case 2: printf("Wednesday"); break;
        case 3: printf("Thursday"); break;
        case 4: printf("Friday"); break;
        case 5: printf("Saturday"); break;
        case 6: printf("Sunday"); break;
    }

    return 0;
}
```

Output



Conclusion

The program correctly computes the weekday for 1-Jan of any given year.

Q5. Using the ternary operator, input the length & breadth of at least 3 rectangles and print which rectangle has the highest perimeter.

Aim

To find which of three rectangles has the largest perimeter using a ternary operator.

Algorithm

1. Input length & breadth of 3 rectangles
2. Compute perimeters
3. Use ternary operator to compare
4. Display rectangle number with highest perimeter

C Program

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

```
int main() {
    int l1,b1,l2,b2,l3,b3;
    int p1, p2, p3;

    printf("Enter length & breadth of rectangle 1: ");
    scanf("%d %d", &l1, &b1);

    printf("Enter length & breadth of rectangle 2: ");
    scanf("%d %d", &l2, &b2);

    printf("Enter length & breadth of rectangle 3: ");
    scanf("%d %d", &l3, &b3);

    p1 = 2 * (l1 + b1);
    p2 = 2 * (l2 + b2);
    p3 = 2 * (l3 + b3);

    int max = (p1 > p2 && p1 > p3) ? 1 :
              (p2 > p3) ? 2 : 3;

    printf("Rectangle %d has the highest perimeter.\n", max);

    return 0;
}
```

Output

.

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp3.1.5.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter length and breadth of Rectangle 1: 4
3
Enter length and breadth of Rectangle 2: 6
5
Enter length and breadth of Rectangle 3: 8
7

Perimeter of Rectangle 1 = 14.00
Perimeter of Rectangle 2 = 22.00
Perimeter of Rectangle 3 = 30.00

Highest Perimeter = 30.00
Rectangle 3 has the highest perimeter.
PS C:\Users\ASUS\Desktop\C Exp>
```

Conclusion

Using the ternary operator simplifies conditional comparisons to find the rectangle with the highest perimeter.

.

EXPERIMENT3.2 – LOOPS

Q1. Program to enter numbers till user wants & display count of +ve, –ve, zero

---

Aim

To repeatedly accept numbers from the user until they choose to stop, and then print the count of positive, negative, and zero values.

Algorithm

1. Start.
2. Initialize counters: positive = 0, negative = 0, zero = 0.
3. Use a loop that continues until the user decides to stop.
4. Input a number.
5. If number > 0 → increment positive.
6. Else if number < 0 → increment negative.
7. Else → increment zero.
8. Ask the user whether they want to continue.
9. End loop when user enters 'n' or 'N'.
10. Display the counts.
11. Stop.

Pseudocode

```
pos = neg = zero = 0
repeat
    input num
    if num > 0 then pos++
    else if num < 0 then neg++
    else zero++
    ask user continue? (y/n)
until user enters n
print pos, neg, zero
```

Flowchart (Text)

```
START
  ↓
Initialize counters
  ↓
Repeat Loop:
  ↓
Input number
  ↓
Check sign (+ / - / 0)
  ↓
Update counter
  ↓
Ask: Continue? y/n
  ↓
If yes → repeat
If no  → exit loop
  ↓
Print result
  ↓
STOP
```

C Program

```
#include <stdio.h>

int main() {
    int num, pos = 0, neg = 0, zero = 0;
    char ch;

    do {
        printf("Enter a number: ");
        scanf("%d", &num);

        if (num > 0) pos++;
        else if (num < 0) neg++;
        else zero++;

        printf("Do you want to continue? (y/n): ");
        scanf(" %c", &ch);

    } while (ch == 'y' || ch == 'Y');

    printf("\nPositive numbers: %d", pos);
    printf("\nNegative numbers: %d", neg);
    printf("\nZeroes: %d\n", zero);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp3.2.1.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter a number: 4
Do you want to enter another number? (y/n): y
Enter a number: -5
Do you want to enter another number? (y/n): y
Enter a number: 0
Do you want to enter another number? (y/n): n

Count of Positive numbers = 1
Count of Negative numbers = 1
Count of Zeros = 1
PS C:\Users\ASUS\Desktop\C Exp>
```

Q2. Print multiplication table of a number

Aim

To print the multiplication table of a number entered by the user.

Algorithm

1. Start.
2. Input the number.
3. Loop i from 1 to 10.
4. Print number × i.
5. Stop.

Pseudocode

```
input n
for i = 1 to 10
    print n × i
```

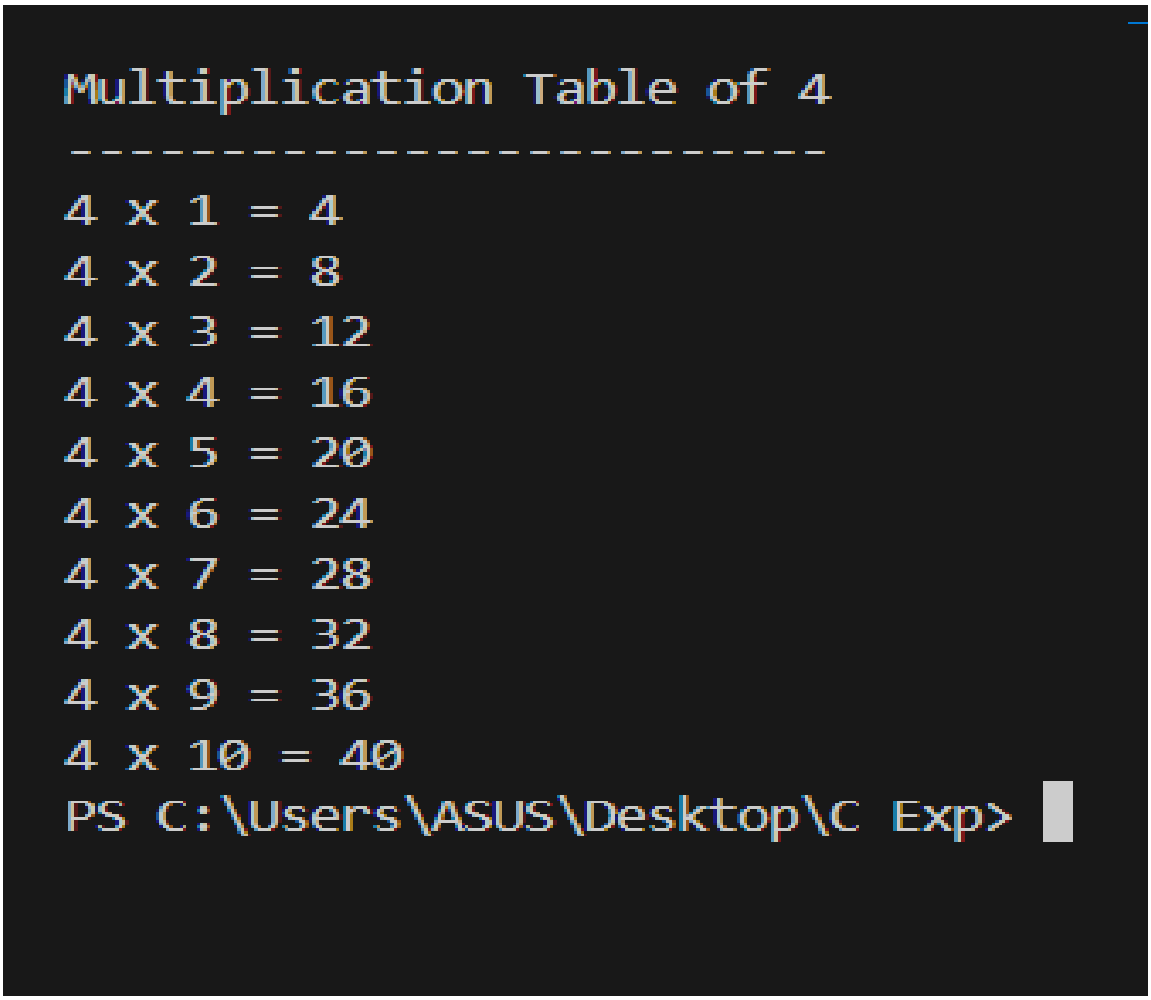
Flowchart

```
START → Input number → Loop i = 1 to 10 → Print n*i → END
```

C Program

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);

    for (int i = 1; i <= 10; i++)
        printf("%d × %d = %d\n", n, i, n * i);

    return 0;
}
```

Output



Q3. Generate the following pattern

a)

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

b)

```
1
12
123
1234
```

Aim

To print custom star and number patterns using loops.

Algorithm

1. Start.
2. Loop through rows.
3. For each row, print required characters.
4. Stop.

Pseudocode

```
for r = 1 to n
    print r stars

for r = 1 to n
    print numbers from 1 to r
```

C Program

```
#include <stdio.h>

int main() {
    int i, j;

    // Pattern A
    for (i = 1; i <= 4; i++) {
```

```
        for (j = 1; j <= i; j++) {
            printf("*");
        }
        printf("\n");
    }

    // Pattern B
    for (i = 1; i <= 4; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d", j);
        }
        printf("\n");
    }

    return 0;
}
```

Output

Q4. Town population problem

Initial population = 100000

Growth rate = 10% per year

Print population at end of each year for last 10 years

Aim

To compute the population of a town after each year using a fixed growth rate.

Algorithm

1. Start.
2. Initialize population = 100000.
3. Loop for 10 years:
    a. population = population + (population × 0.10)
    b. print population
4. Stop.

Pseudocode

```
pop = 100000
for year = 1 to 10
    pop = pop + 0.10 * pop
    print pop
```

C Program

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

```c
int main() {
    float pop = 100000;

    for (int year = 1; year <= 10; year++) {
        pop = pop + (0.10 * pop);
        printf("Year %d Population: %.2f\n", year, pop);
    }

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp3.2.4.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Year    Population
---------------------
1       110000.00
2       121000.00
3       133100.00
4       146410.00
5       161051.00
6       177156.09
7       194871.70
8       214358.88
9       235794.77
10      259374.25
PS C:\Users\ASUS\Desktop\C Exp>
```

Q5. Ramanujan Numbers up to a limit

A Ramanujan number satisfies:

$a^3 + b^3 = c^3 + d^3$ (a,b,c,d > 0)

Aim

To print all Ramanujan (taxicab) numbers up to a given limit.

Algorithm

1. Start.
2. Input limit.
3. Use 4 nested loops to test combinations.
4. If $a^3 + b^3 == c^3 + d^3$ → print number.
5. Stop.

Pseudocode

```
input limit
for a=1 to limit
 for b=a to limit
  for c=1 to limit
   for d=c to limit
     if (a^3 + b^3 == c^3 + d^3)
         print number
```

C Program

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

```c
int main() {
    int limit;
    printf("Enter limit: ");
    scanf("%d", &limit);

    for (int a = 1; a <= limit; a++)
    for (int b = a; b <= limit; b++)
    for (int c = 1; c <= limit; c++)
    for (int d = c; d <= limit; d++) {
        int s1 = a*a*a + b*b*b;
        int s2 = c*c*c + d*d*d;

        if (s1 == s2 && a != c && a != d) {
            printf("%d = %d^3 + %d^3 = %d^3 + %d^3\n",
                   s1, a, b, c, d);
        }
    }
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp3.2.5.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter limit: 2000

Ramanujan Numbers up to 2000 are:
--------------------------------
1729 = 1^3 + 12^3 = 9^3 + 10^3
PS C:\Users\ASUS\Desktop\C Exp>
```

EXPERIMENT4: VARIABLES AND SCOPE OF VARIABLES

Q1. Declare a global variable outside all functions & use it in various functions

---

Aim

To understand how a global variable can be accessed inside multiple functions.

Theory

A **global variable** is declared outside all functions.

It can be accessed by **all functions** in the same program.

Algorithm

1. Declare a global variable.
2. Create multiple functions.
3. Access and modify the global variable inside each function.
4. Observe output.
5. End.

Pseudocode

```
declare global variable x
function A → print x
function B → modify x
main → call A and B
```

Flowchart

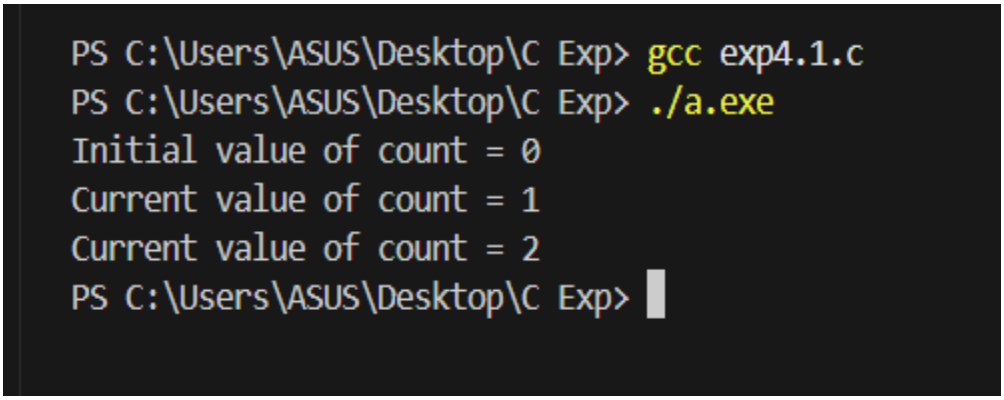
```
START
 ↓
Declare global variable
 ↓
Call function A → prints global variable
 ↓
Call function B → modifies global variable
 ↓
END
```

C Program

```
#include <stdio.h>

int globalVar = 10;   // Global variable

void func1() {
    printf("Inside func1: globalVar = %d\n", globalVar);
}

void func2() {
    globalVar += 5;
    printf("Inside func2: globalVar = %d\n", globalVar);
}

int main() {
    printf("Inside main: globalVar = %d\n", globalVar);
```

```
    func1();
    func2();

    printf("Back in main: globalVar = %d\n", globalVar);
    return 0;
}
```

Output



Q2. Declare a local variable inside a function & try accessing it outside

---

Aim

To compare accessibility of local vs global variables.

Theory

- A **local variable** exists only inside its function.
- It cannot be accessed from other functions or main().
- Global variables can be accessed anywhere.

Algorithm

1. Create function with local variable.
2. Try to access it in main.
3. Observe compile-time error.
4. End.

Pseudocode

```
function test:
    declare local x = 5
    print x
main:
    try print x → ERROR
```

Flowchart

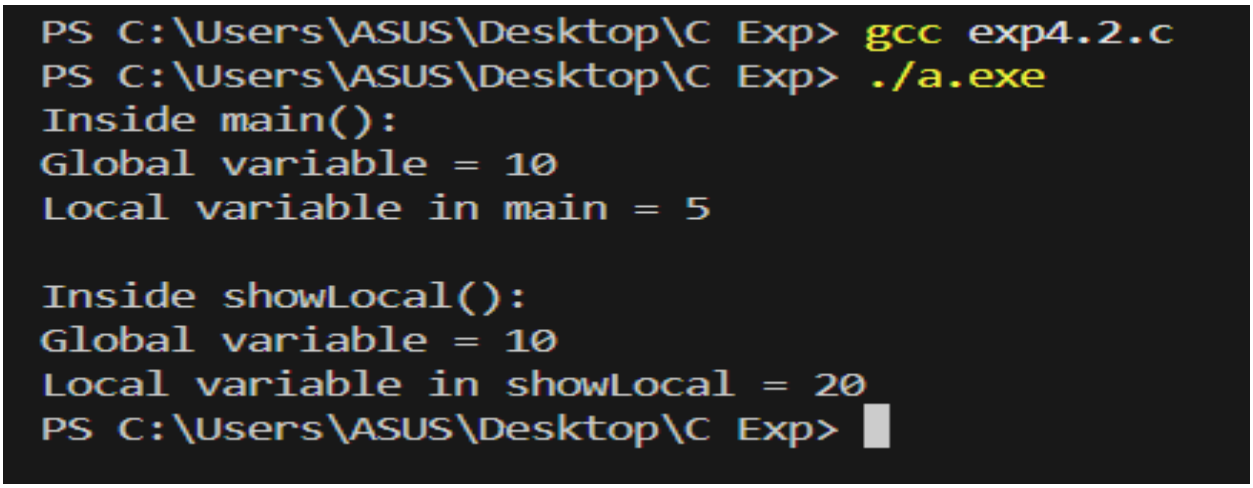
```
Function block creates local variable
 ↓
Local variable destroyed after function ends
 ↓
Access outside → ERROR
```

C Program

```
#include <stdio.h>

void test() {
    int localVar = 20;   // Local variable
```

```
    printf("Inside test(): localVar = %d\n", localVar);
}

int main() {
    test();

    // printf("%d", localVar);  // ERROR: localVar is not accessible
here

    printf("Cannot access localVar outside test() function.\n");
    return 0;
}
```

Output



Q3. Declare variables inside different code blocks and test accessibility

---

Aim

To understand block-level scope.

Theory

A **block** is a section enclosed in { }.

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

Algorithm

1. Create blocks using { }.
2. Declare variables inside each block.
3. Try accessing them outside.
4. Observe errors.
5. End.

Pseudocode

```
main:
  start block1
     declare a
     print a
  end block1
```

```
  try print a → ERROR
```

Flowchart

```
START
↓
Enter Block
↓
Block variable created
↓
Exit Block
↓
Block variable destroyed
↓
Access outside → ERROR
```

C Program

```
#include <stdio.h>

int main() {
    {
        int x = 100;
        printf("Inside block 1: x = %d\n", x);
    }

    // printf("%d", x); // ERROR: x not accessible

    {
        int y = 200;
        printf("Inside block 2: y = %d\n", y);
    }

    // printf("%d", y); // ERROR: y not accessible

    printf("Variables inside blocks cannot be accessed outside the
```

```
block.\n");
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp4.3.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Outside inner block: x = 10
Inside inner block: x = 10, y = 20
Inside nested block: x = 10, y = 20, z = 30

Back to main block: x = 10
PS C:\Users\ASUS\Desktop\C Exp>
```

Q4. Declare a static local variable & observe persistence across calls

---

Aim

To study persistence of static local variables across function calls.

Theory

- A **static local variable** retains its value between function calls.
- Unlike normal local variables, static variables are initialized only once.

Algorithm

1. Create function with static variable.
2. Call function repeatedly.
3. Observe value incrementing.
4. End.

Pseudocode

```
function test:
    static x = 0
    x++
    print x

main:
    call test 3 times
```

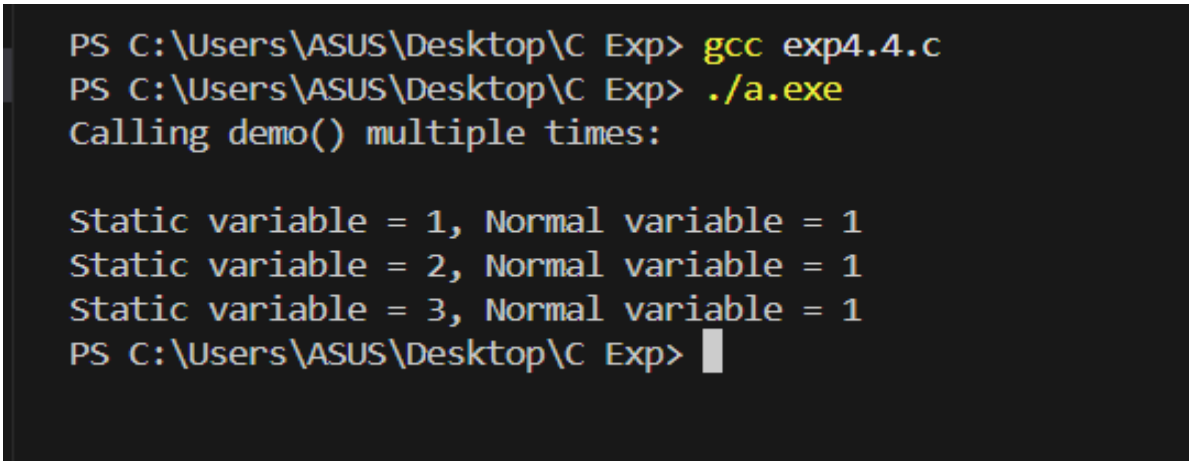
Flowchart

```
START
↓
Function creates static variable (one-time)
↓
Function call → updated value
↓
Function call → retains previous value
↓
Function call → retains again
↓
END
```

C Program

```c
#include <stdio.h>

void display() {
    static int count = 0;  // Static local variable
    count++;
    printf("Function called %d times\n", count);
}

int main() {
    display();
    display();
    display();
    return 0;
}
```

Output



EXPERIMENT 5 – ARRAYS

Q1. Read a list of integers into a 1-D array & print the second largest number

Aim

To find the **second largest element** in a list of integers.

Algorithm

1. Start
2. Input number of elements
3. Read all elements into array
4. Find largest and second largest
5. Print second largest
6. End

Pseudocode

```
read n
read array A[n]
largest = second = -∞
for i in 0..n-1:
    if A[i] > largest:
        second = largest
        largest = A[i]
    else if A[i] > second AND A[i] != largest:
        second = A[i]
```

```
print second
```

Flowchart

```
START
 ↓
Read n
 ↓
Read array
 ↓
Find largest & second largest
 ↓
Print second largest
 ↓
END
```

C Program

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

int main() {
    int n, i, arr[100];
    int largest = INT_MIN, second = INT_MIN;

    printf("Enter number of integers: ");
    scanf("%d", &n);

    printf("Enter %d integers:\n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for(i = 0; i < n; i++) {
        if(arr[i] > largest) {
            second = largest;
            largest = arr[i];
```

```
        } else if(arr[i] > second && arr[i] != largest) {
            second = arr[i];
        }
    }

    printf("Second largest = %d\n", second);
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\C Exp> gcc exp5.1.c
PS C:\Users\ASUS\Desktop\C Exp> ./a.exe
Enter number of elements: 5
Enter 5 integers:
1
3
4
5
6
The second largest number is: 5
PS C:\Users\ASUS\Desktop\C Exp>
```

Q2. Count positive, negative, odd & even numbers in an array

Aim

To count **positive**, **negative**, **odd**, and **even** integers in a 1D array.

Algorithm

1. Read n
2. Read array
3. Initialize count variables
4. Traverse array and update counts
5. Display counts

Pseudocode

```
read n
read array A[n]

pos=neg=odd=even=0

for each element:
    if >0 pos++
    else if <0 neg++
    if %2==0 even++
    else odd++

print pos,neg,odd,even
```

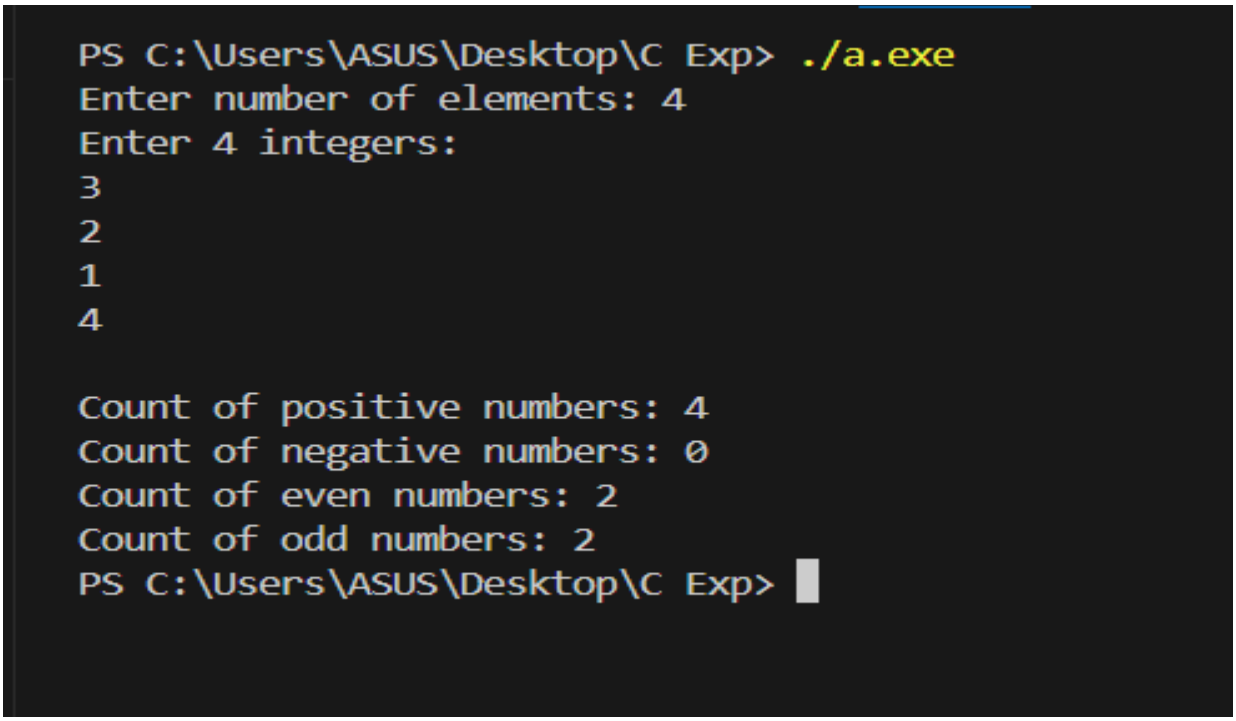
Flowchart

```
START → Read array → For each element:
      → Update pos/neg/odd/even → Print results → END
```

C Program

```
#include <stdio.h>

int main() {
    int n, i, arr[100];
    int pos = 0, neg = 0, odd = 0, even = 0;

    printf("Enter number of integers: ");
    scanf("%d", &n);

    printf("Enter %d integers:\n", n);
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    for(i = 0; i < n; i++) {
        if(arr[i] > 0) pos++;
        else if(arr[i] < 0) neg++;

        if(arr[i] % 2 == 0) even++;
        else odd++;
    }

    printf("Positive = %d\nNegative = %d\nOdd = %d\nEven = %d\n",
           pos, neg, odd, even);

    return 0;
}
```

Output



Q3. Find the frequency of a particular number in an array

Aim

To count how many times a given number appears in the array.

Algorithm

1. Read n

2. Read array
3. Input element to search
4. Loop through array counting matches
5. Print frequency

Pseudocode

```
read n
read array
read key
count=0
for each element:
    if element == key:
        count++
print count
```

Flowchart

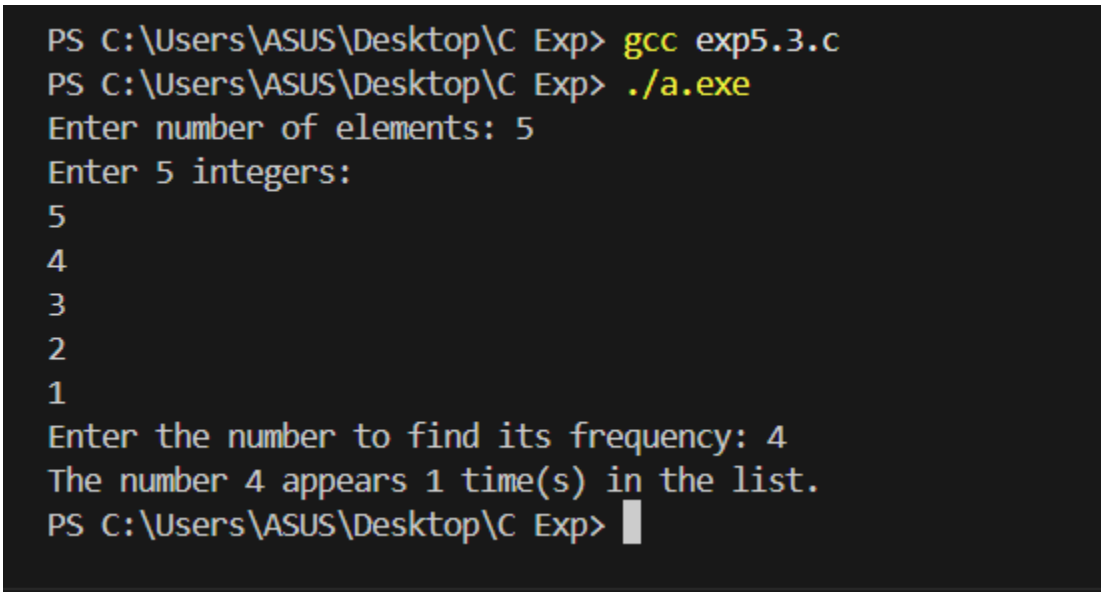
```
START → Read array → Input key
 → Count occurrences → Print frequency → END
```

C Program

```
#include <stdio.h>

int main() {
    int n, i, arr[100], key, count = 0;

    printf("Enter number of integers: ");
    scanf("%d", &n);

    printf("Enter integers:\n");
    for(i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter number to find frequency: ");
```

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

    for(i = 0; i < n; i++)
        if(arr[i] == key)
            count++;

    printf("Frequency of %d is %d\n", key, count);

    return 0;
}
```

Output



Q4. Matrix Multiplication (Check compatibility + Print matrices)

Aim

To multiply two matrices A(m×n) and B(p×q) and check if multiplication is possible.

Algorithm

1. Input rows & columns of A and B
2. If n ≠ p → multiplication not possible
3. Else
    a. Read matrices A and B
    b. Multiply using formula
    c. Print matrices

Pseudocode

```
read m,n
read matrix A
read p,q
if n != p → print incompatible
else
   for i in m
     for j in q
        C[i][j] = sum(A[i][k] * B[k][j])
print C
```

Flowchart

```
START
 ↓
Read size of A and B
 ↓
Check if n == p
 ↓Yes
Compute product
 ↓
Print A, B, C
```

↓
END

(No → Print “Incompatible”)

C Program

```
#include <stdio.h>

int main() {
    int m, n, p, q, i, j, k;
    int A[10][10], B[10][10], C[10][10];

    printf("Enter rows and columns of Matrix A: ");
    scanf("%d %d", &m, &n);

    printf("Enter elements of Matrix A:\n");
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            scanf("%d", &A[i][j]);

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

    if(n != p) {
        printf("Matrix multiplication NOT possible!\n");
        return 0;
    }

    printf("Enter elements of Matrix B:\n");
    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] += A[i][k] * B[k][j];
        }

    printf("\nMatrix A:\n");
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++)
            printf("%d ", A[i][j]);
        printf("\n");
    }

    printf("\nMatrix B:\n");
    for(i = 0; i < p; i++) {
        for(j = 0; j < q; j++)
            printf("%d ", B[i][j]);
        printf("\n");
    }

    printf("\nProduct Matrix C = A × B:\n");
    for(i = 0; i < m; i++) {
        for(j = 0; j < q; j++)
            printf("%d ", C[i][j]);
        printf("\n");
    }

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter rows and columns of Matrix A: 2
2
Enter elements of Matrix A:
2
2
2
3
Enter rows and columns of Matrix B: 2
2
Enter elements of Matrix B:
2
3
4
5

Matrix A:
2 2
2 3

Matrix B:
2 3
4 5

Product Matrix C = A ├ù B:
12 16
16 21
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

.

EXPERIMENT6: FUNCTIONS

Experiment 1: Factorial (Recursive & Non-Recursive) + Binomial Coefficient

Aim

To develop recursive and non-recursive functions to compute factorial of a number and use them to calculate the binomial coefficient for different values of *n* and *r*.

Algorithm

Factorial (Recursive)

1. If n = 0, return 1
2. Else return n × FACT(n−1)

Factorial (Non-Recursive)

1. Set fact = 1
2. Run loop i = 1 to n
3. fact = fact × i
4. Return fact

Binomial Coefficient

1. Accept n, r
2. Use formula:

[

nCr = \frac{n!}{r!(n-r)!}

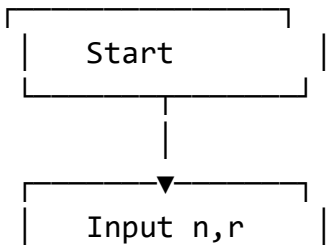
]

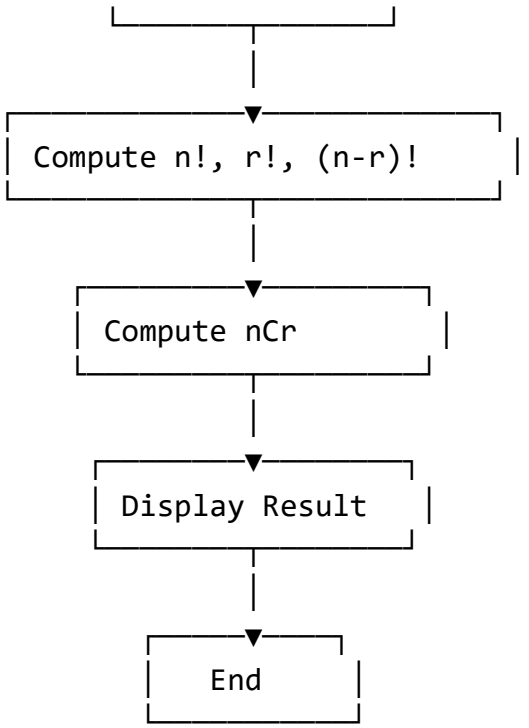
3. Display result

Pseudocode

```
FUNCTION factorial_rec(n)
    IF n == 0 THEN
        RETURN 1
    ELSE
        RETURN n * factorial_rec(n-1)
END FUNCTION
FUNCTION factorial_nonrec(n)
    fact = 1
    FOR i = 1 TO n
        fact = fact * i
    END FOR
    RETURN fact
END FUNCTION
READ n, r
nr = factorial(n) / (factorial(r) * factorial(n-r))
DISPLAY nr
```

Flowchart (ASCII)



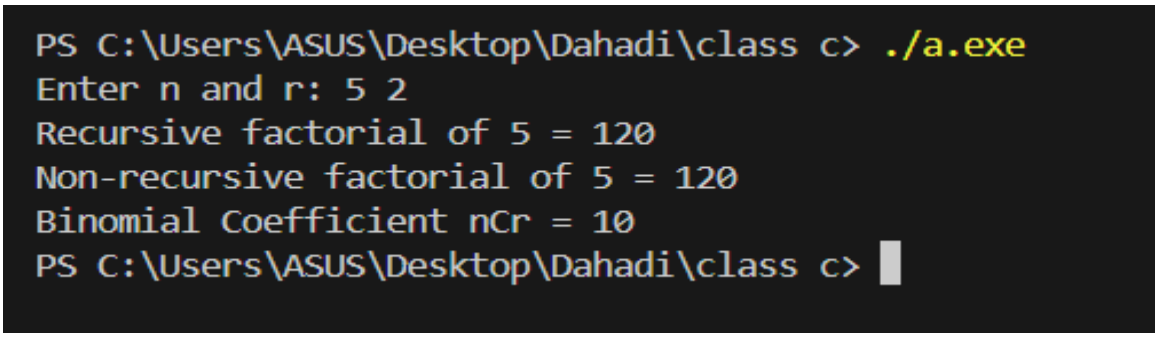


C Program

```
#include <stdio.h>

// Recursive factorial
int fact_rec(int n) {
    if (n == 0) return 1;
    return n * fact_rec(n - 1);
}

// Non-recursive factorial
int fact_nonrec(int n) {
    int fact = 1;
    for (int i = 1; i <= n; i++)
        fact *= i;
    return fact;
}
```

```
int main() {
    int n, r;

    printf("Enter n and r: ");
    scanf("%d %d", &n, &r);

    int fr = fact_rec(n);
    int fn = fact_nonrec(n);

    int nCr = fact_rec(n) / (fact_rec(r) * fact_rec(n - r));

    printf("Recursive factorial of %d = %d\n", n, fr);
    printf("Non-recursive factorial of %d = %d\n", n, fn);
    printf("Binomial Coefficient nCr = %d\n", nCr);

    return 0;
}
```

Output



Conclusion

Both recursive and non-recursive functions correctly compute factorial. Using factorial values, the binomial coefficient was successfully calculated.

Experiment 2: Recursive GCD Function

Aim

To develop a recursive function to find the GCD of two integers.

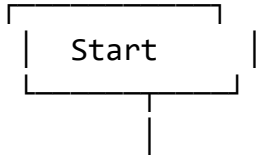
Algorithm

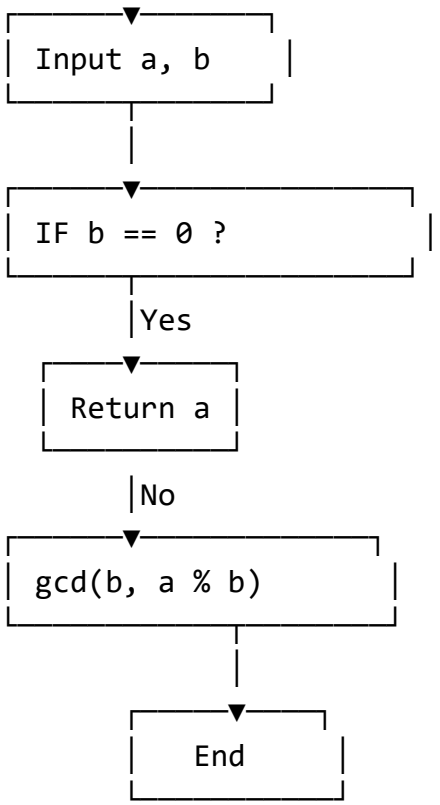
1. If b = 0 return a
2. Else return GCD(b, a % b)

Pseudocode

```
FUNCTION gcd(a, b)
    IF b == 0 THEN
        RETURN a
    ELSE
        RETURN gcd(b, a % b)
END FUNCTION
```

Flowchart (ASCII)





C Program

```
#include <stdio.h>

int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

int main() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    printf("GCD = %d\n", gcd(a, b));
    return 0;
```

```
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc fun1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter two integers: 6
8
GCD = 2
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

The recursive function successfully computes the greatest common divisor using Euclid's algorithm.

Experiment 3: Recursive Fibonacci Function

Aim

To develop a recursive function to generate the Fibonacci series up to n.

Algorithm

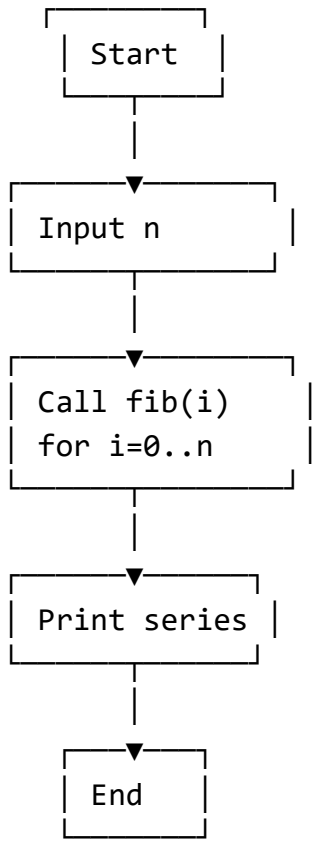
1. If n == 0 return 0
2. If n == 1 return 1

3. Else return F(n−1) + F(n−2)

Pseudocode

```
FUNCTION fib(n)
    IF n == 0 RETURN 0
    IF n == 1 RETURN 1
    RETURN fib(n-1) + fib(n-2)
END FUNCTION
```

Flowchart (ASCII)



C Program

```c
#include <stdio.h>

int fib(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}

int main() {
    int n;
    printf("Enter limit: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        printf("%d ", fib(i));

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc fun1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter limit: 4
0 1 1 2
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

The Fibonacci series was successfully generated using recursive function calls.

Experiment 4: Prime Number Function

Aim

To develop a function ISPRIME(num) to test if a number is prime and print all primes in a given range.

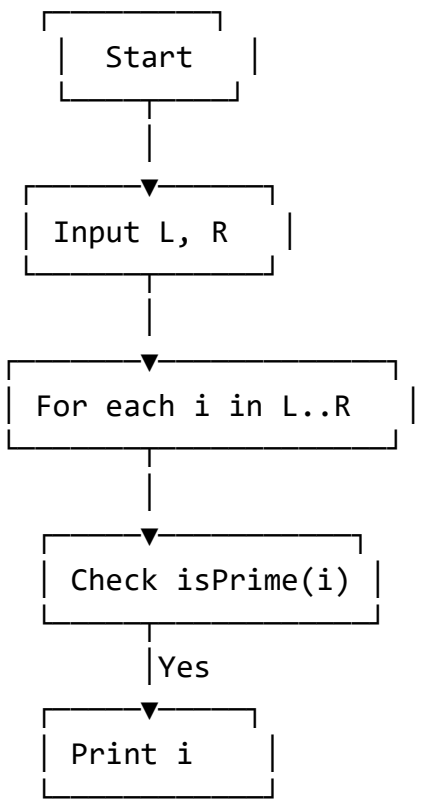
Algorithm

1. If n < 2 return false
2. Check divisibility from 2 to $\sqrt{n}$
3. If divisible → not prime
4. Else prime

Pseudocode

```
FUNCTION isPrime(n)
    IF n < 2 RETURN 0
    FOR i = 2 TO sqrt(n)
        IF n % i == 0 RETURN 0
    RETURN 1
END FUNCTION
```

Flowchart (ASCII)

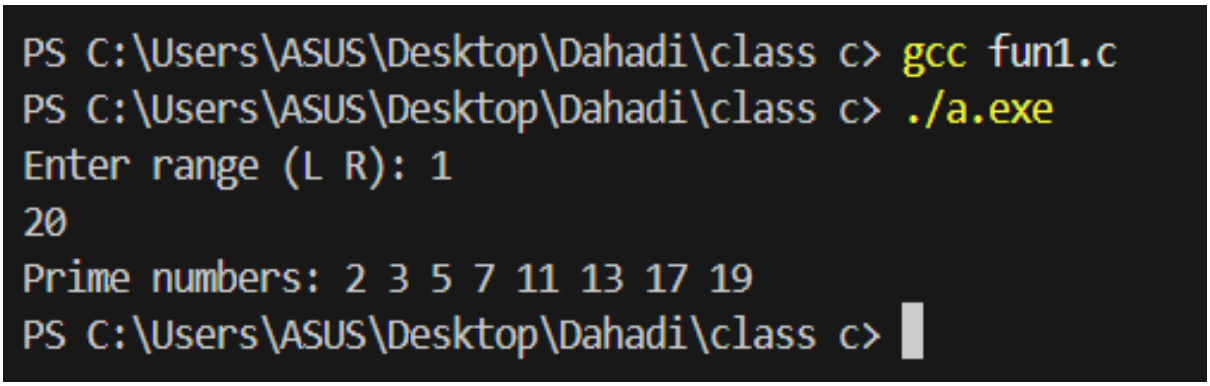


C Program

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

int isPrime(int n) {
    if (n < 2) return 0;
    for (int i = 2; i <= sqrt(n); i++)
        if (n % i == 0) return 0;
    return 1;
}

int main() {
    int L, R;
```

```
    printf("Enter range (L R): ");
    scanf("%d %d", &L, &R);

    printf("Prime numbers: ");
    for (int i = L; i <= R; i++)
        if (isPrime(i))
            printf("%d ", i);

    return 0;
}
```

Output



Conclusion

The program successfully identifies prime numbers within the specified range.

Experiment 5: Reverse a String

Aim

To develop a function that reverses a string using character processing.

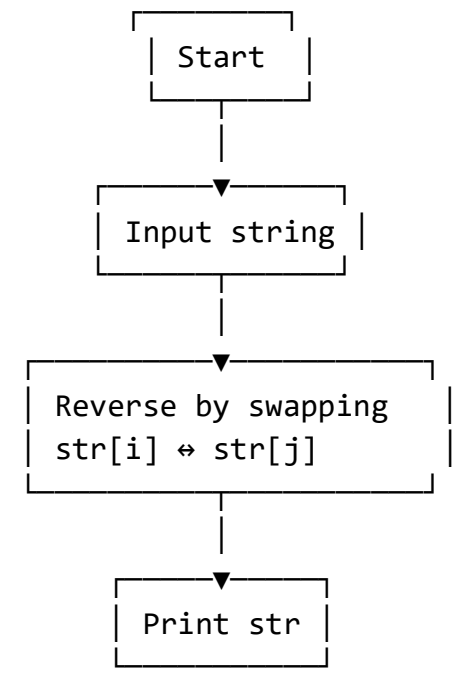
Algorithm

1. Read input string
2. Find length
3. Swap characters from both ends
4. Print reversed string

Pseudocode

```
FUNCTION reverse(str)
    i = 0
    j = length(str) - 1
    WHILE i < j
        swap str[i] and str[j]
        i++
        j--
END FUNCTION
```

Flowchart (ASCII)



C Program

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

void reverse(char str[]) {
    int i = 0;
    int j = strlen(str) - 1;

    while (i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}
```

```c
int main() {
    char str[50];

    printf("Enter a string: ");
    scanf("%s", str);

    reverse(str);

    printf("Reversed string = %s\n", str);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc fun1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter a string: Omakreshwar
Reversed string = rawhserkamO
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

The reverse() function works correctly by swapping characters from both ends.

EXPERIMENT7: STRUCTURES AND UNION

Experiment 1: Complex Number using Structure

Aim

To perform reading, writing, addition, and subtraction of two complex numbers using structures and functions.

Algorithm

1. Define structure with real and imaginary parts
2. Read two complex numbers
3. Add and subtract them
4. Display results

Pseudocode

```
STRUCT Complex
    real, imag
END STRUCT
```

```
FUNCTION readComplex()
FUNCTION printComplex()
FUNCTION addComplex()
FUNCTION subComplex()
```

Flowchart (ASCII)

```
Start
   ↓
Read Complex Numbers
   ↓
Add / Subtract
   ↓
Display Result
   ↓
 End
```

C Program

```
#include <stdio.h>

struct Complex {
    float real, imag;
};

struct Complex readComplex() {
    struct Complex c;
scanf("%f %f", &c.real, &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 c;
    c.real = a.real + b.real;
    c.imag = a.imag + b.imag;
    return c;
}

struct Complex sub(struct Complex a, struct Complex b) {
    struct Complex c;
    c.real = a.real - b.real;
    c.imag = a.imag - b.imag;
    return c;
}

int main() {
    struct Complex c1, c2, sum, diff;

    printf("Enter first complex number (real imag): ");
    c1 = readComplex();

    printf("Enter second complex number (real imag): ");
    c2 = readComplex();

    sum = add(c1, c2);
    diff = sub(c1, c2);

    printf("Sum = ");
    printComplex(sum);

    printf("Difference = ");
    printComplex(diff);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc struct.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter first complex number (real imag): 4 5
Enter second complex number (real imag): 64
32
Sum = 68.00 + 37.00i
Difference = -60.00 + -27.00i
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Structures and functions simplify complex number operations.

Experiment 2: Employee Salary Using Structure

Aim

To calculate and display employee gross salary using structure.

Algorithm

1. Read name and basic pay
2. Calculate DA = 10% of basic
3. Gross = Basic + DA
4. Print name and gross salary

Pseudocode

```
READ name, basic
DA = basic * 0.1
gross = basic + DA
DISPLAY name, gross
```

Flowchart (ASCII)

```
Start
   ↓
Read Data
   ↓
Calculate DA & Gross
   ↓
Display
   ↓
 End
```

C Program

```
#include <stdio.h>

struct Employee {
    char name[30];
    float basic, da, gross;
};

int main() {
    struct Employee e;
```

```
    printf("Enter name: ");
    scanf("%s", e.name);

    printf("Enter basic pay: ");
    scanf("%f", &e.basic);

    e.da = e.basic * 0.10;
    e.gross = e.basic + e.da;

    printf("Employee Name: %s\n", e.name);
    printf("Gross Salary: %.2f\n", e.gross);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc struct.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter name: Om
Enter basic pay: 2000000
Employee Name: Om
Gross Salary: 2200000.00
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Employee salary calculation was successfully implemented using structures.

Experiment 3: Book Structure Passed to Function

Aim

To pass a structure to a function and display book details.

Algorithm

1. Read book details
2. Pass structure to function
3. Print details

Pseudocode

```
STRUCT Book
READ book details
CALL printBook(book)
```

Flowchart (ASCII)

```
Start
   ↓
Read Book Data
   ↓
Pass to Function
   ↓
```

```
Display Details
   ↓
 End
```

C Program

```
#include <stdio.h>

struct Book {
    int id;
    char title[30];
    char author[30];
    float price;
};

void printBook(struct Book b) {
    printf("ID: %d\nTitle: %s\nAuthor: %s\nPrice: %.2f\n",
           b.id, b.title, b.author, b.price);
}

int main() {
    struct Book b;

    printf("Enter ID, Title, Author, Price: ");
    scanf("%d %s %s %f", &b.id, b.title, b.author, &b.price);

    printBook(b);
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc struct.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter ID, Title, Author, Price: 4179
Harry Potter
JK Rowling
ID: 4179
Title: Harry
Author: Potter
Price: 0.00
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Structures can be easily passed to functions for modular programming.

Experiment 4: Union for Address Storage

Aim

To demonstrate the use of union for storing address details.

Algorithm

1. Define union with address fields
2. Read present address
3. Display it

Pseudocode

```
UNION Address
READ address
DISPLAY address
```

Flowchart (ASCII)

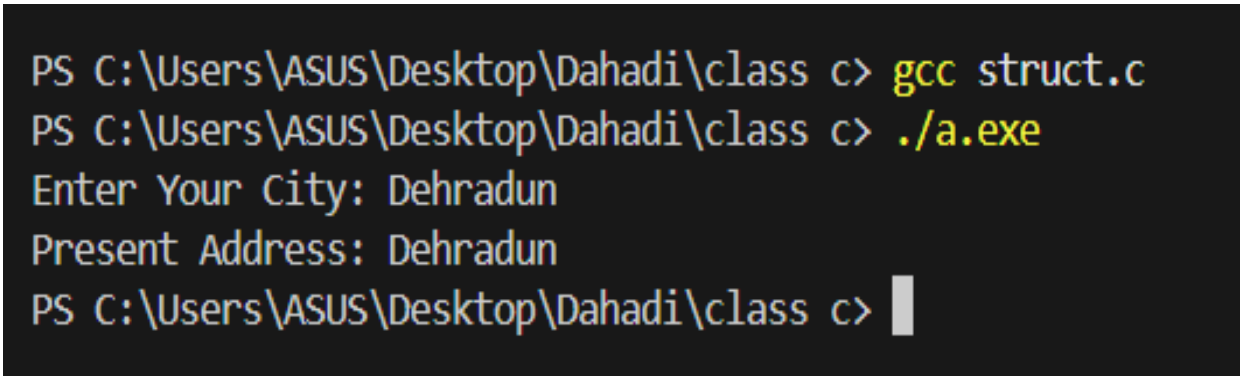
```
Start
   ↓
Input Address
   ↓
Display Address
   ↓
 End
```

C Program

```
#include <stdio.h>

union Address {
    char home[50];
    char hostel[50];
    char city[20];
};

int main() {
    union Address addr;

    printf("Enter Your City: ");
    scanf("%s", addr.city);
```

```
    printf("Present Address: %s\n", addr.city);

    return 0;
}
```

Output



Conclusion

Union efficiently stores different data types using shared memory.

EXPERIMENT8: POINTERS

Experiment 1: Declaration and Initialization of Pointers

Aim

To declare and initialize pointers of different data types and display their addresses and values.

Algorithm

1. Declare int, float, and char variables
2. Declare pointers for each data type
3. Assign addresses of variables to pointers
4. Print variable values, pointer values, and addresses

Pseudocode

```
DECLARE int a, float b, char c
DECLARE int*, float*, char*
ASSIGN addresses of variables to pointers
PRINT variable values
```

```
PRINT pointer values and dereferenced values
```

Flowchart (ASCII)

```
Start
   ↓
Declare Variables
   ↓
Declare Pointers
   ↓
Assign Addresses
   ↓
Display Values
   ↓
 End
```

C Program

```c
#include <stdio.h>

int main() {
    int a = 10;
    float b = 3.14;
    char c = 'A';

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

    printf("Integer value: %d, Address: %p\n", a, p1);
    printf("Float value: %.2f, Address: %p\n", b, p2);
    printf("Char value: %c, Address: %p\n", c, p3);
```

```
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc point1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Integer value: 10, Address: 0061FF10
Float value: 3.14, Address: 0061FF0C
Char value: A, Address: 0061FF0B
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Pointers successfully store addresses of variables and allow indirect access to their values.

Experiment 2: Pointer Arithmetic

Aim

To perform increment and decrement operations on pointers and observe memory address changes.

Algorithm

1. Declare integer array
2. Assign base address to pointer
3. Increment and decrement pointer
4. Display addresses and values

Pseudocode

```
DECLARE array
DECLARE pointer
POINT to array[0]
INCREMENT pointer
DECREMENT pointer
DISPLAY addresses and values
```

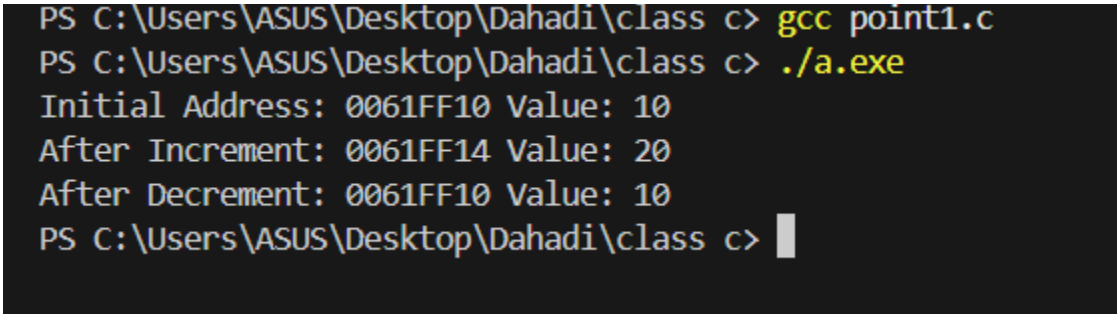
Flowchart (ASCII)

```
Start
   ↓
Declare Array
   ↓
Assign Pointer
   ↓
Increment / Decrement Pointer
   ↓
Display Values
   ↓
 End
```

C Program

```c
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30};
    int *p = arr;

    printf("Initial Address: %p Value: %d\n", p, *p);

    p++;
    printf("After Increment: %p Value: %d\n", p, *p);

    p--;
    printf("After Decrement: %p Value: %d\n", p, *p);

    return 0;
}
```

Output



Conclusion

Pointer arithmetic changes the address based on the size of the data type (e.g., 4 bytes for int).

Experiment 3: Passing Variables by Reference Using Pointers

Aim

To modify variable values by passing them to a function using pointers.

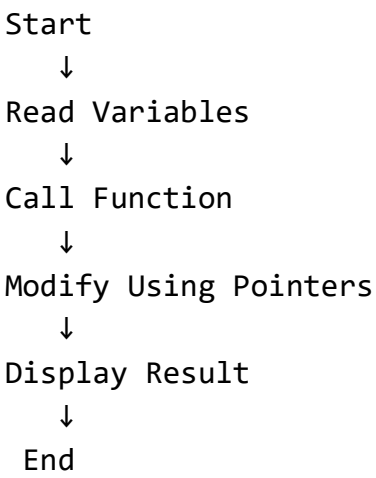
Algorithm

1. Declare variables
2. Pass their addresses to a function
3. Modify values inside the function
4. Display modified values

Pseudocode

```
FUNCTION modify(x, y)
    *x = *x + 10
    *y = *y * 2
END FUNCTION

MAIN
    PASS addresses to function
    DISPLAY modified values
```

Flowchart (ASCII)



C Program

```c
#include <stdio.h>

void modify(int *x, int *y) {
    *x = *x + 10;
    *y = *y * 2;
}

int main() {
    int a = 5, b = 4;

    printf("Before: a=%d b=%d\n", a, b);
    modify(&a, &b);
    printf("After: a=%d b=%d\n", a, b);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc point1.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Before: a=5 b=4
After: a=15 b=8
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Passing variables by reference using pointers allows functions to modify original values.

EXPERIMENT9: FILE HANDLING IN C

Experiment 1: Creating a File and Writing Text

Aim

To create a new file and write text into it using file handling functions in C.

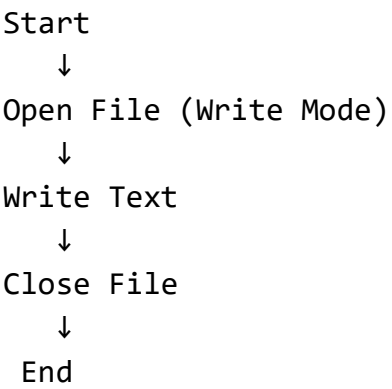
Algorithm

1. Declare a file pointer
2. Open file in write mode
3. Write text using `fprintf()`
4. Close file

Pseudocode

```
DECLARE file pointer
OPEN file in write mode
WRITE text into file
CLOSE file
```

Flowchart (ASCII)



C Program

```c
#include <stdio.h>

int main() {
    FILE *fp;

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

    if (fp == NULL) {
        printf("File cannot be created\n");
        return 1;
    }

    fprintf(fp, "Welcome to File Handling in C.\n");
    fprintf(fp, "This is a sample file.\n");

    fclose(fp);

    printf("File created and data written successfully.\n");
    return 0;
```

```
}
```

Output

```
Before: a=5 b=4
After: a=15 b=8
PS C:\Users\ASUS\Desktop\Dahadi\class c>
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc file.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
File created and data written successfully.
PS C:\Users\ASUS\Desktop\Dahadi\class c> ls
```

```
-a----         12/7/2025  10:17 PM             56 sample.txt
```

Conclusion

The program successfully created a new file and wrote text into it using file handling functions.

Experiment 2: Reading File Content Character by Character

Aim

To read the contents of an existing file character by character and display it on the console.

Algorithm

1. Open file in read mode
2. Read characters using `fgetc()`
3. Display each character
4. Close file

Pseudocode

```
OPEN file in read mode
WHILE character != EOF
    READ character
    DISPLAY character
CLOSE file
```

Flowchart (ASCII)

```
Start
   ↓
Open File (Read Mode)
   ↓
Read Character
   ↓
Is EOF?
   ↓ No
Display Character
   ↓
Repeat
   ↓ Yes
Close File
   ↓
```

End

C Program

```
#include <stdio.h>

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

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

    if (fp == NULL) {
        printf("File cannot be opened\n");
        return 1;
    }

    printf("File Contents:\n");
    while ((ch = fgetc(fp)) != EOF) {
        putchar(ch);
    }

    fclose(fp);
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc file.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
File Contents:
Welcome to File Handling in C.
This is a sample file.
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

The program read and displayed file contents character by character successfully.

Experiment 3: Reading File Content Line by Line

Aim

To read the contents of a file line by line and display each line on the console.

Algorithm

1. Open file in read mode
2. Read lines using `fgets()`
3. Display each line

4. Close file

Pseudocode

```
OPEN file in read mode
WHILE fgets reads a line
    DISPLAY line
CLOSE file
```

Flowchart (ASCII)

```
Start
   ↓
Open File
   ↓
Read Line
   ↓
Is EOF?
   ↓ No
Display Line
   ↓
Repeat
   ↓ Yes
Close File
   ↓
 End
```

C Program

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

```c
int main() {
    FILE *fp;
    char line[100];

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

    if (fp == NULL) {
        printf("File cannot be opened\n");
        return 1;
    }

    printf("File Contents (Line by Line):\n");

    while (fgets(line, sizeof(line), fp) != NULL) {
        printf("%s", line);
    }

    fclose(fp);
    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc file.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
File Contents (Line by Line):
Welcome to File Handling in C.
This is a sample file.
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

The program successfully read and displayed file contents line by line using file handling functions in C.

EXPERIMENT10: DYNAMIC MEMORY ALLOCATION

Experiment 1: Creation of a Simple Linked List

Aim

To create a simple singly linked list using dynamic memory allocation with pointers and structures.

Algorithm

1. Define a node structure containing data and next pointer
2. Use `malloc()` to create nodes dynamically
3. Link nodes using pointers
4. Display the linked list

Pseudocode

```
STRUCT Node
    data
    next
END STRUCT

CREATE nodes using malloc
```

```
LINK nodes
DISPLAY list
```

Flowchart (ASCII)

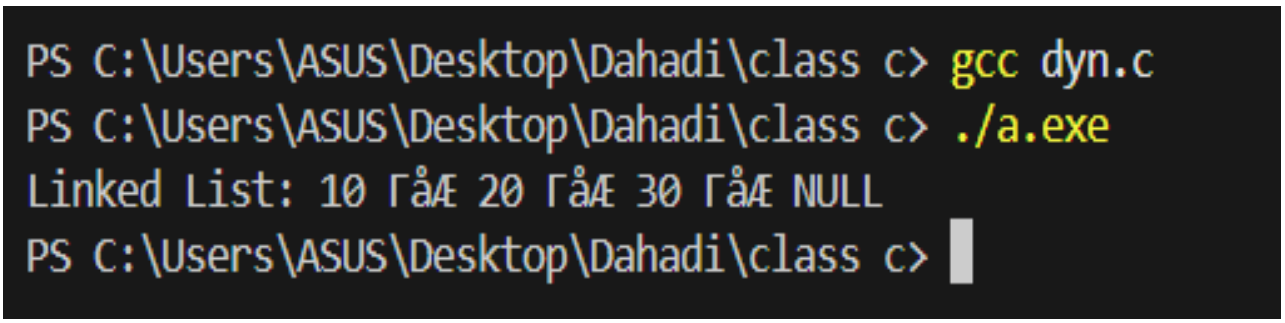
```
Start
   ↓
Create First Node
   ↓
Create Next Node
   ↓
Link Nodes
   ↓
Display List
   ↓
 End
```

C Program

```
#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;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d → ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");

    return 0;
}
```

Output



Conclusion

A singly linked list was successfully created using dynamic memory allocation and pointers.

Experiment 2: Insertion in the Middle of a Linked List

Aim

To insert a new element at the middle position of a singly linked list.

Algorithm

1. Create initial linked list
2. Find middle position
3. Create a new node using `malloc()`
4. Insert the new node by adjusting pointers
5. Display updated list

Pseudocode

```
CREATE linked list
CALCULATE length
mid = length / 2
MOVE pointer to mid-1
```

```
INSERT new node
DISPLAY list
```

Flowchart (ASCII)

```
Start
   ↓
Create Linked List
   ↓
Find Middle
   ↓
Create New Node
   ↓
Insert Node
   ↓
Display List
   ↓
 End
```

C Program

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

struct Node {
    int data;
    struct Node *next;
};

void display(struct Node *head) {
    while (head != NULL) {
        printf("%d → ", head->data);
        head = head->next;
```

```
    }
    printf("NULL\n");
}

int main() {
    struct Node *head, *second, *third, *newNode;
    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;

    printf("Original List: ");
    display(head);

    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = 25;

    newNode->next = second->next;
    second->next = newNode;

    printf("After Insertion in Middle: ");
    display(head);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc dyn.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Original List: 10 ΓåÆ 20 ΓåÆ 30 ΓåÆ NULL
After Insertion in Middle: 10 ΓåÆ 20 ΓåÆ 25 ΓåÆ 30 ΓåÆ NULL
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Insertion in the middle of a linked list was successfully implemented using dynamic memory allocation and pointer manipulation.

EXPERIMENT11: BITWISE OPERATORS

Experiment 1 Bitwise AND, OR, NOT Operators

Aim

To perform bitwise AND, OR, and NOT operations on integer values.

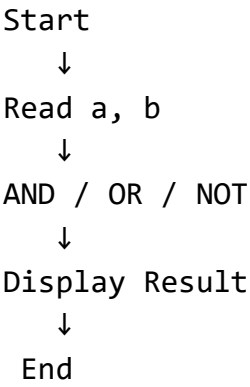
Algorithm

1. Read two integers
2. Apply bitwise AND
3. Apply bitwise OR
4. Apply bitwise NOT
5. Display results

Pseudocode

```
READ a, b
AND = a & b
OR = a | b
NOT a = ~a
DISPLAY results
```

Flowchart (ASCII)



C Program

```c
#include <stdio.h>

int main() {
    int a, b;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    printf("Bitwise AND (a & b): %d\n", a & b);
    printf("Bitwise OR  (a | b): %d\n", a | b);
    printf("Bitwise NOT (~a): %d\n", ~a);

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc t.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter two integers: 3
2
Bitwise AND (a & b): 2
Bitwise OR  (a | b): 3
Bitwise NOT (~a): -4
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Bitwise operators perform operations at the bit level and are useful for low-level programming.

Experiment 2: Left Shift and Right Shift Operators

Aim

To demonstrate left shift and right shift bitwise operators in C.

Algorithm

1. Read an integer
2. Apply left shift operator
3. Apply right shift operator
4. Display the results

Pseudocode

```
READ num
LEFT = num << 1
RIGHT = num >> 1
DISPLAY results
```

Flowchart (ASCII)

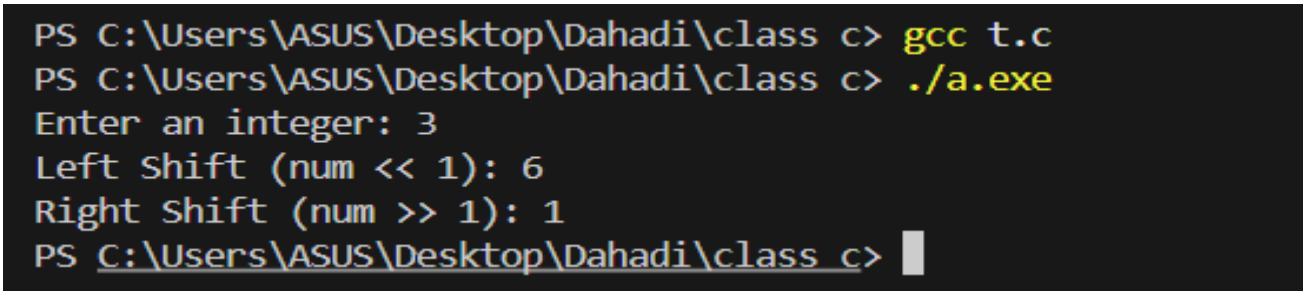
```
Start
   ↓
Read Number
   ↓
Left Shift / Right Shift
   ↓
Display Result
   ↓
 End
```

C Program

```
#include <stdio.h>

int main() {
```

```
    int num;

    printf("Enter an integer: ");
    scanf("%d", &num);

    printf("Left Shift (num << 1): %d\n", num << 1);
    printf("Right Shift (num >> 1): %d\n", num >> 1);

    return 0;
}
```

Output



Conclusion

Shift operators move bits left or right, effectively multiplying or dividing the number by powers of 2.

EXPERIMENT12: PREPROCESSOR AND DIRECTIVES IN C

Experiment 1: Defining Constants Using Preprocessor Directives

Aim

To define and use constant variables in C using preprocessor directives.

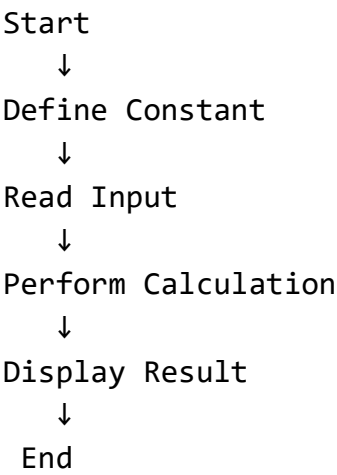
Algorithm

1. Use `#define` to declare constants
2. Read required input values
3. Perform calculations using constants
4. Display results

Pseudocode

```
DEFINE PI = 3.14
READ radius
area = PI * radius * radius

DISPLAY area
```

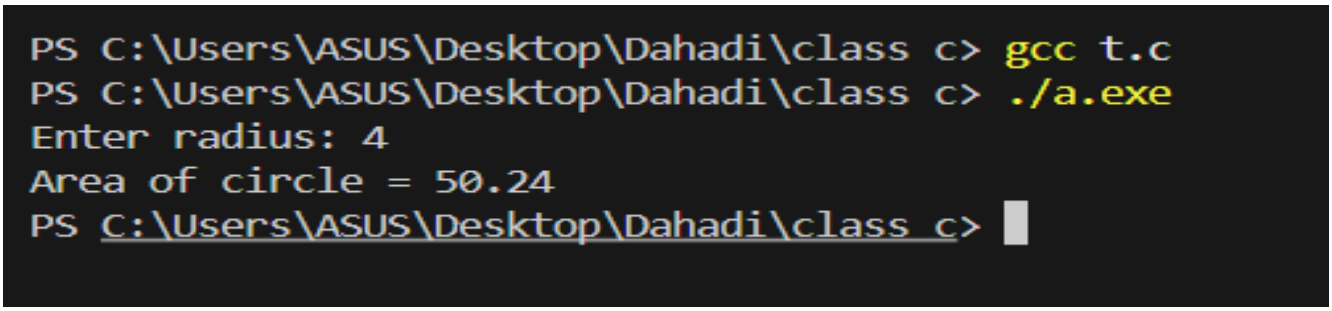
Flowchart (ASCII)



C Program

```c
#include <stdio.h>

#define PI 3.14

int main() {
    float r, area;

    printf("Enter radius: ");
    scanf("%f", &r);

    area = PI * r * r;

    printf("Area of circle = %.2f\n", area);
    return 0;
}
```

Output



Conclusion

Preprocessor directive `#define` was successfully used to declare constants in the program.

Experiment 2 : Defining Function Using Preprocessor Directives

Aim

To define and use a macro function using preprocessor directives in C.

Algorithm

1. Define macro function using `#define`
2. Read input numbers
3. Apply macro function
4. Display result

Pseudocode

```
DEFINE MAX(a, b)
READ a, b
result = MAX(a, b)
DISPLAY result
```

Flowchart (ASCII)

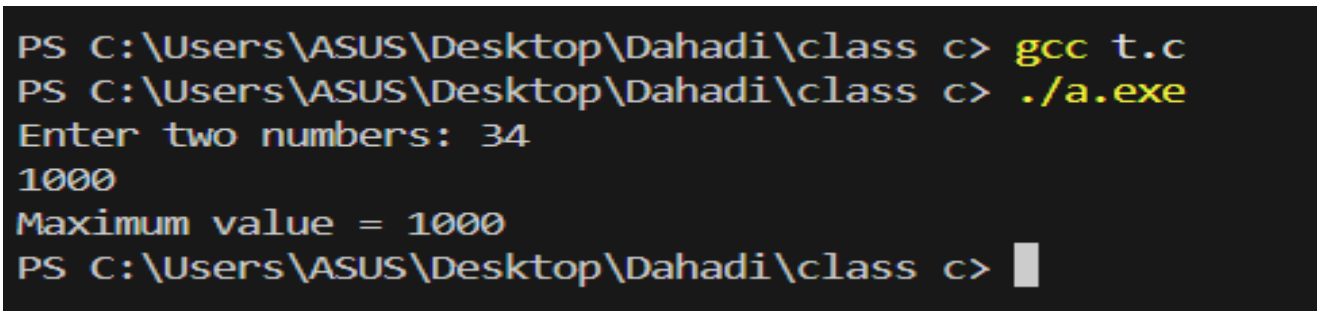
```
Start
   ↓
Define Macro Function
   ↓
Read Inputs
   ↓
Apply Macro
   ↓
Display Result
   ↓
 End
```

C Program

```
#include <stdio.h>

#define MAX(a, b) ((a > b) ? a : b)

int main() {
    int x, y;

    printf("Enter two numbers: ");
```

```
    scanf("%d %d", &x, &y);

    printf("Maximum value = %d\n", MAX(x, y));

    return 0;
}
```

Output



Conclusion

Macro functions defined using preprocessor directives execute faster as they avoid function call overhead.

EXPERIMENT13: MACROS IN C

Experiment 1: Arithmetic Operations Using Macros

Aim

To define and use multiple macros in C to perform basic arithmetic operations.

Algorithm

1. Define macros for addition, subtraction, multiplication, and division
2. Read two numbers
3. Apply macro operations
4. Display results

Pseudocode

```
DEFINE ADD(a,b)
DEFINE SUB(a,b)
DEFINE MUL(a,b)
DEFINE DIV(a,b)

READ a, b
```

```
DISPLAY ADD, SUB, MUL, DIV
```

Flowchart (ASCII)

```
Start
   ↓
Define Macros
   ↓
Read Two Numbers
   ↓
Apply Macros
   ↓
Display Results
   ↓
 End
```

C Program

```
#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, y;

    printf("Enter two integers: ");
    scanf("%d %d", &x, &y);

    printf("Addition = %d\n", ADD(x, y));
    printf("Subtraction = %d\n", SUB(x, y));
```

```
    printf("Multiplication = %d\n", MUL(x, y));

    if (y != 0)
        printf("Division = %d\n", DIV(x, y));
    else
        printf("Division by zero not allowed\n");

    return 0;
}
```

Output

```
PS C:\Users\ASUS\Desktop\Dahadi\class c> gcc t.c
PS C:\Users\ASUS\Desktop\Dahadi\class c> ./a.exe
Enter two integers: 2
3
Addition = 5
Subtraction = -1
Multiplication = 6
Division = 0
PS C:\Users\ASUS\Desktop\Dahadi\class c>
```

Conclusion

Macros allow fast execution of arithmetic operations by avoiding function call overhead. Proper use of parentheses prevents logical errors.