Programming Lab Practical File

Name	Sujal Singh
Roll Number	
Branch	IIOT–B1–23
School	University School of Automation & Robotics
University	Guru Gobind Singh Indraprastha University

Index

Experiment–0

Aim:

Write a C program to print “Hello, World!”

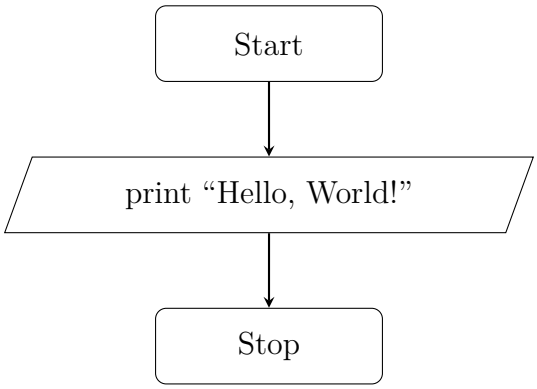
Description:

Program to print “Hello, World”.

Algorithm:

1: Start
2: **print** “Hello, World!”
3: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    printf("Hello, World!");  // prints "Hello, World!"
    return 0;
}
```

Input & Output:

```
Hello, World!
```

Experiment–1

Aim:

Write a C program to find the greatest number among three numbers provided by the user using if else.

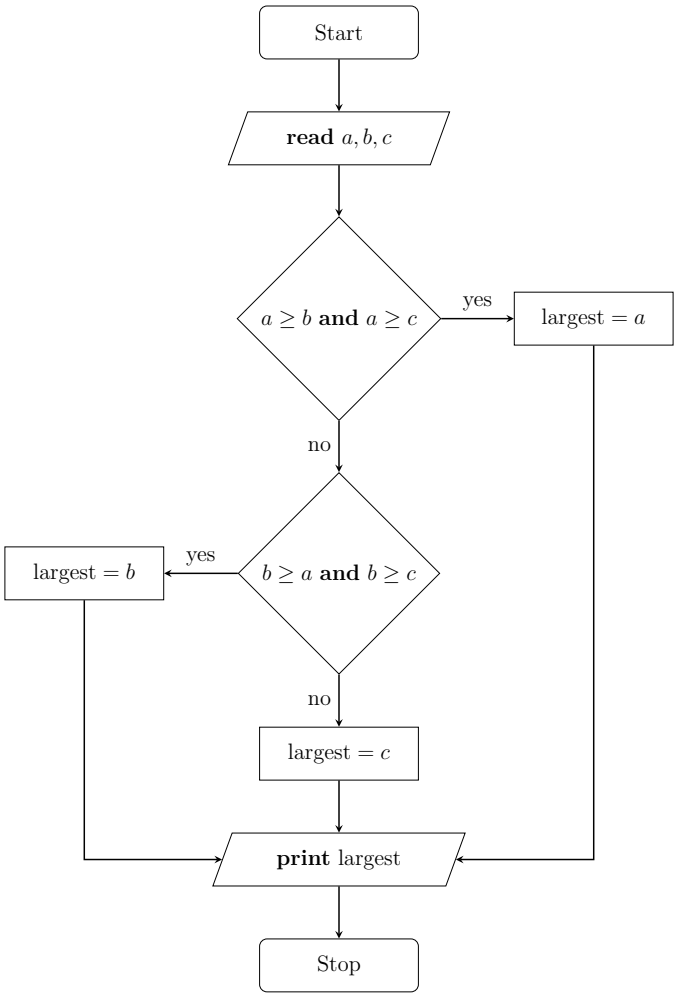
Description:

Program that finds the largest number among three numbers input by the user.

Algorithm:

1: Start
2: **read** a, b, c
3: **if** $a \geq b$ **and** $a \geq c$ **then**
4: | largest $\leftarrow a$
5: **else if** $b \geq a$ **and** $b \geq c$ **then**
6: | largest $\leftarrow b$
7: **else**
8: | largest $\leftarrow c$
9: **print** largest
10: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    int a, b, c, largest;

    // Read first number
    printf("Enter first number: ");
    scanf("%d", &a);

    // Read second number
    printf("Enter second number: ");
    scanf("%d", &b);

    // Read third number
    printf("Enter third number: ");
    scanf("%d", &c);

    if (a >= b && a >= c) {
        // a is larger than both b and c
        // (it might also be equal to one of them or even both)
        largest = a;
    } else if (b >= a && b >= c) {
        // b is larger than both a and c
        // (it might also be equal to one of them or even both)
        largest = b;
    } else {
        // Since a and b aren't the largest, c must be.
        largest = c;
    }

    // Output the largest number among the three.
    printf("%d is the largest number.", largest);
    return 0;
}
```

Input & Output:

```
Enter first number: 1
Enter second number: 2
Enter third number: 3
3 is the largest number.
```

Experiment–2

Aim:

Write a C program to find the sum of individual digits of a positive integer using while.

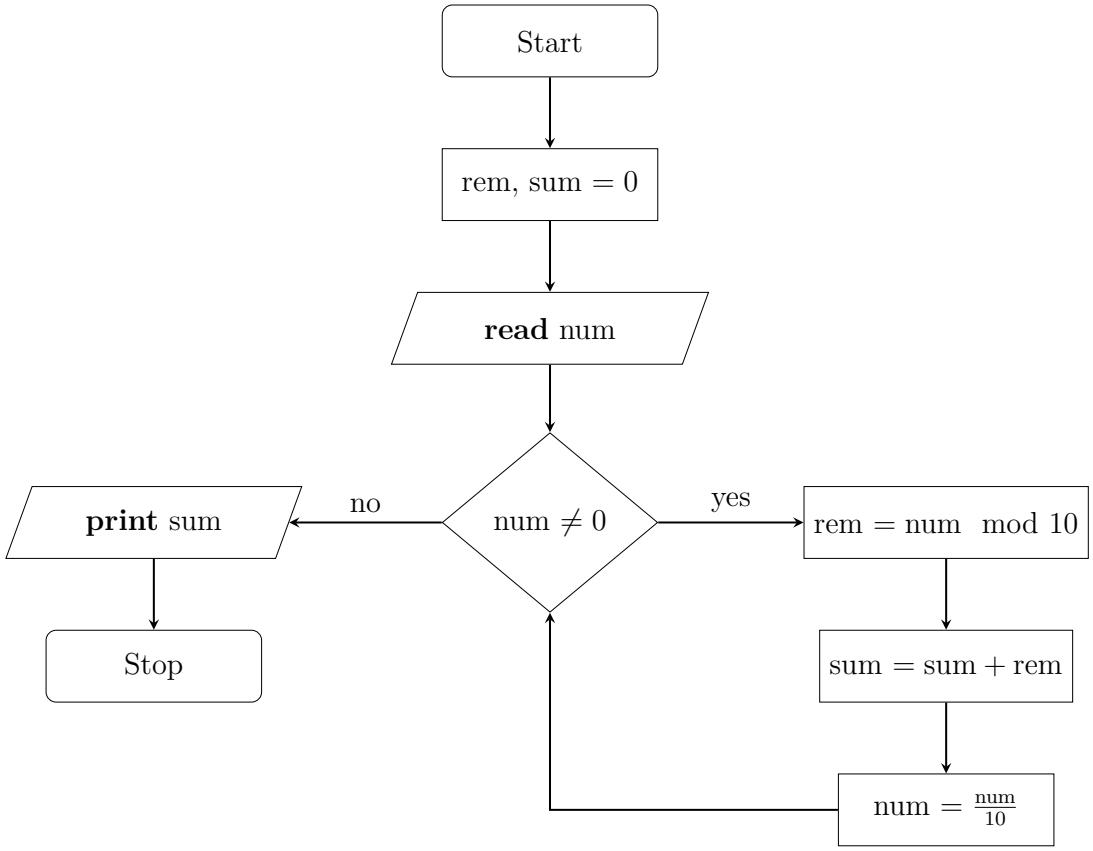
Description:

Program that uses a while loop to find the sum of the individual digits in a number input by the user.

Algorithm:

1: Start
2: rem, sum $\leftarrow 0$
3: **read** num
4: **while** num $\neq 0$ **do**
5: rem $\leftarrow$ num mod 10
6: sum $\leftarrow$ sum + rem
7: num $\leftarrow \frac{\text{num}}{10}$
8: **print** sum
9: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    int num, rem, sum = 0;

    // Read num
    printf("Enter a number: ");
    scanf("%d", &num);

    // Keep removing the one's place from num and add the removed digit to
    ↪ the running sum.
    while (num != 0) {
        rem = num % 10;
        sum += rem;
        num /= 10;
    }

    // Output sum of digits
    printf("Sum of digits = %d", sum);
    return 0;
}
```

Input & Output:

```
Enter a number: 1234
Sum of digits = 10
```

Experiment–3

Aim:

Write a C program to find the roots of a quadratic equation.

Description:

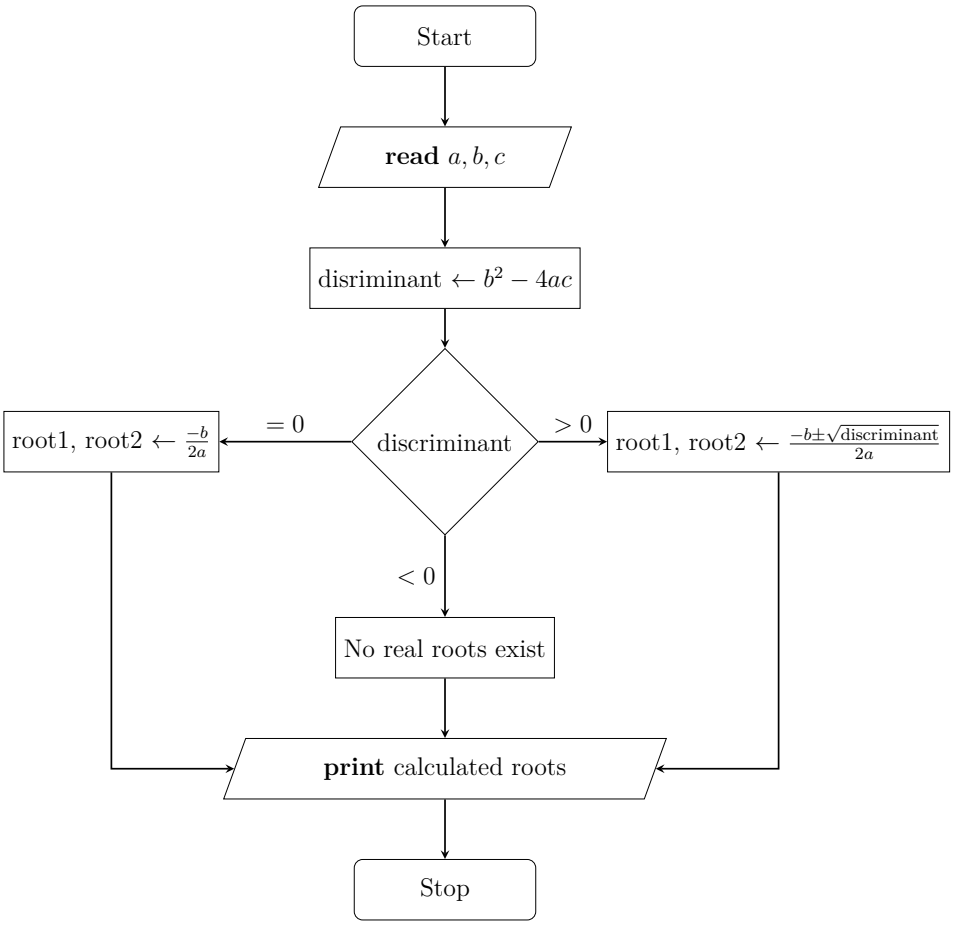
Program that finds the roots of a quadratic equation by using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Algorithm:

1: Start
2: **read** a, b, c
3: discriminant $\leftarrow b^2 - 4ac$
4: **if** discriminant > 0 **then**
5: | root1, root2 $\leftarrow \frac{-b \pm \sqrt{\text{discriminant}}}{2a}$
6: **else if** discriminant $= 0$ **then**
7: | root1, root2 $\leftarrow \frac{-b}{2a}$
8: **else**
9: | No real roots exist
10: **print** root1, root2
11: Stop

Flowchart:



Program:

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

int main() {
    float a, b, c, discriminant, root1, root2;

    // Read a
    printf("Enter x^2 coefficient (a): ");
    scanf("%f", &a);

    // Read b
    printf("Enter x coefficient (b): ");
    scanf("%f", &b);

    // Read c
    printf("Enter constant (c): ");
    scanf("%f", &c);

    // Calculate the discriminant for the given equation
    discriminant = (b * b) - (4 * a * c);

    // Calculate roots by using the quadratic formula
    if (discriminant > 0) {
        root1 = (b + sqrt(discriminant) / (-2 * a));
        root2 = (b - sqrt(discriminant) / (-2 * a));

        printf("Two distinct real roots exist: \n%f \n%f", root1, root2);
    } else if (discriminant == 0) {
        root1 = b / (-2 * a);
        printf("One distinct real root exists: \n%f", root1);
    } else {
        printf("No real roots exist.");
    }

    return 0;
}
```

Input & Output:

```
Enter x^2 coefficient (a): 1
Enter x coefficient (b): 2
Enter constant (c): 3
No real roots exist.
```

Experiment–4

Aim:

Write a C program to perform arithmetic operations using switch case statement

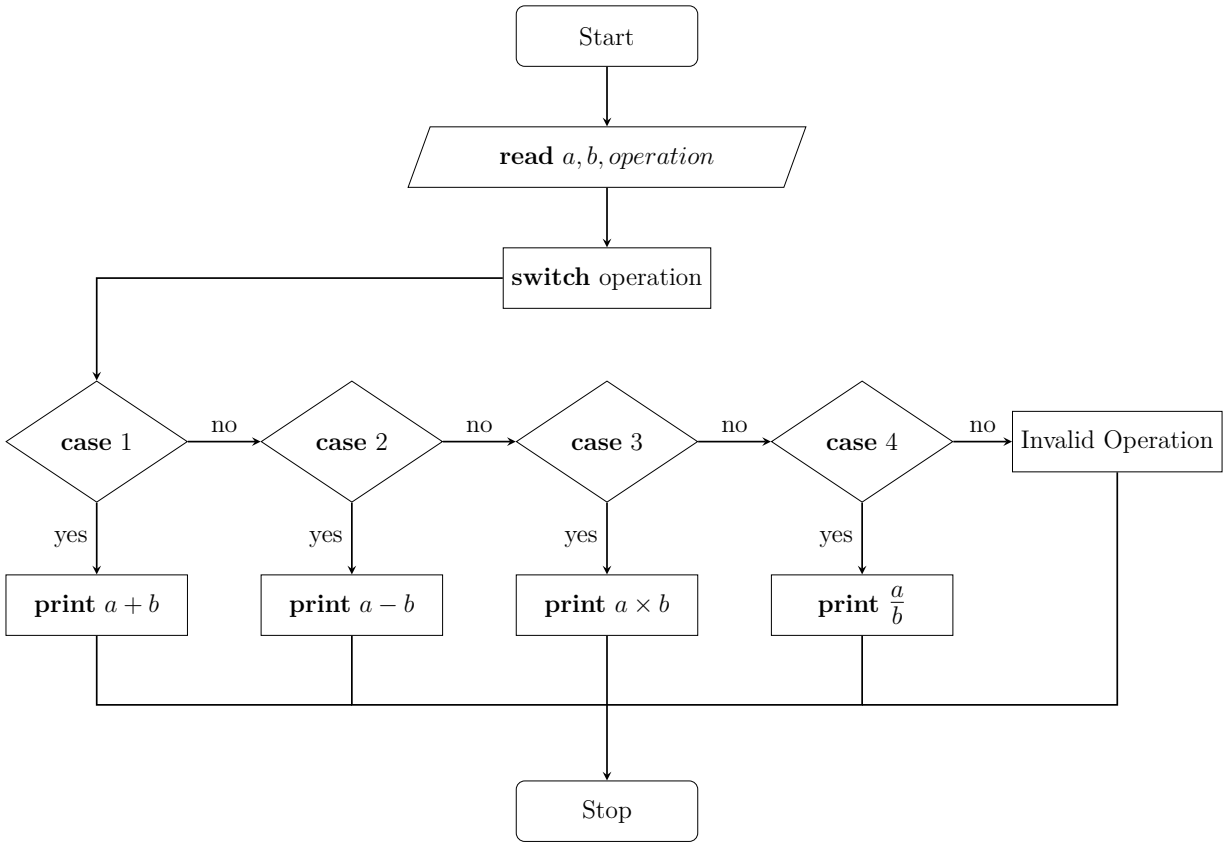
Description:

Program that uses switch case statements to perform an arithmetic operation on two numbers input by the user.

Algorithm:

1: Start
2: **read** a, b, operation
3: **if** operation is add **then**
4: | **print** $a + b$
5: **else if** operation is subtract **then**
6: | **print** $a - b$
7: **else if** operation is multiply **then**
8: | **print** $a \times b$
9: **else if** operation is divide **then**
10: | **print** $\frac{a}{b}$
11: **else**
12: | **print** "Invalid Operation"
13: Stop

Flowchart:



Program:

```c
#include <stdio.h>

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

    // Read a
    printf("Enter first number: ");
    scanf("%d", &a);

    // Read b
    printf("Enter second number: ");
    scanf("%d", &b);

    // Read operation
    printf("Enter operation add(1), subtract(2), multiply(3), divide(4): ");
    scanf("%d", &operation);

    // Perform the operation input by the user on a and b
    switch (operation) {
        case 1:
            printf("%d + %d = %d", a, b, a + b);
            break;
        case 2:
            printf("%d - %d = %d", a, b, a - b);
            break;
        case 3:
            printf("%d x %d = %d", a, b, a * b);
            break;
        case 4:
            printf("%d / %d = %f", a, b, (float) a / b);
            break;
        default:
            printf("Invalid operation.");
    }

    return 0;
}
```

Input & Output:

```
Enter first number: 2
Enter second number: 2
Enter operation add(1), subtract(2), multiply(3), divide(4): 1
2 + 2 = 4
```

Experiment–5(a)

Aim:

Write a C program to find the factorial of a given integer using non-recursive function.

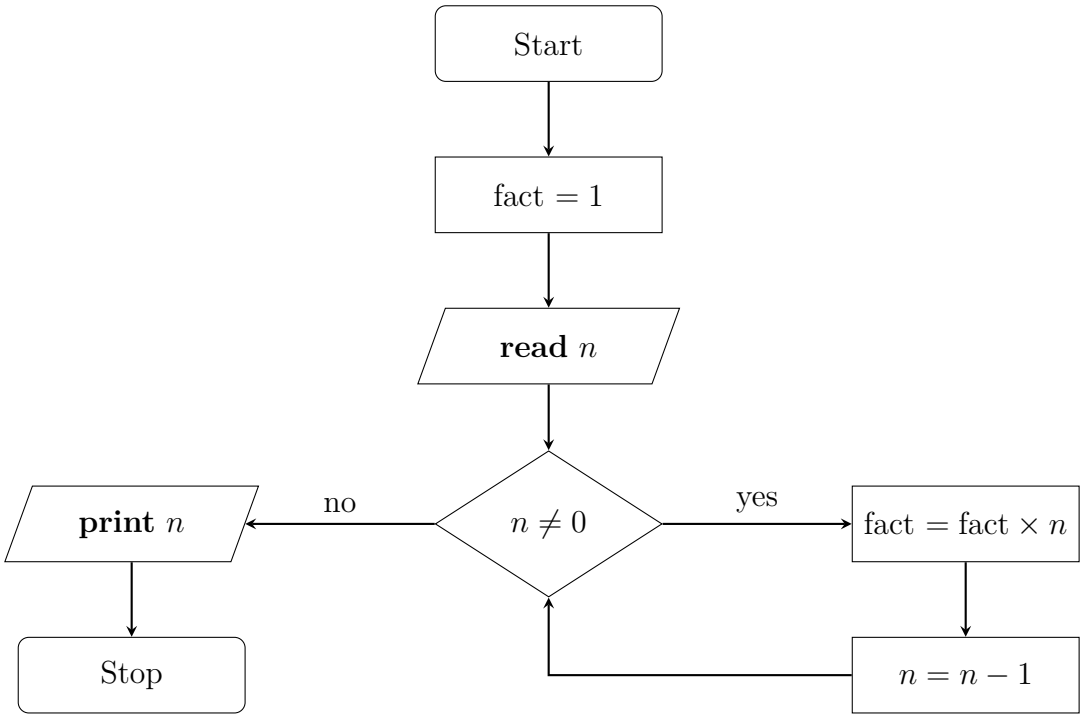
Description:

Program that finds the factorial of a number input by the user without using a recursive function.

Algorithm:

1: Start
2: fact $\leftarrow 1$
3: **read** n
4: **while** $n \neq 0$ **do**
5: $\quad$ fact = fact $\times$ n
6: $\quad$ $n = n - 1$
7: **print** n
8: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    // Initialize fact to 1 since 0! = 1
    int fact = 1, n;

    // Read n
    printf("Enter number: ");
    scanf("%d", &n);

    // Keep multiplying fact by the current value of n and subtract 1 from
    ↪ n on each iteration.
    while (n != 0) {
        fact *= n;
        n -= 1;
    }

    // Output factorial
    printf("Factorial is %d", fact);
    return 0;
}
```

Input & Output:

```
Enter number: 5
Factorial is 120
```

Experiment–5(b)

Aim:

Write a C program to find the factorial of a given integer using recursive function.

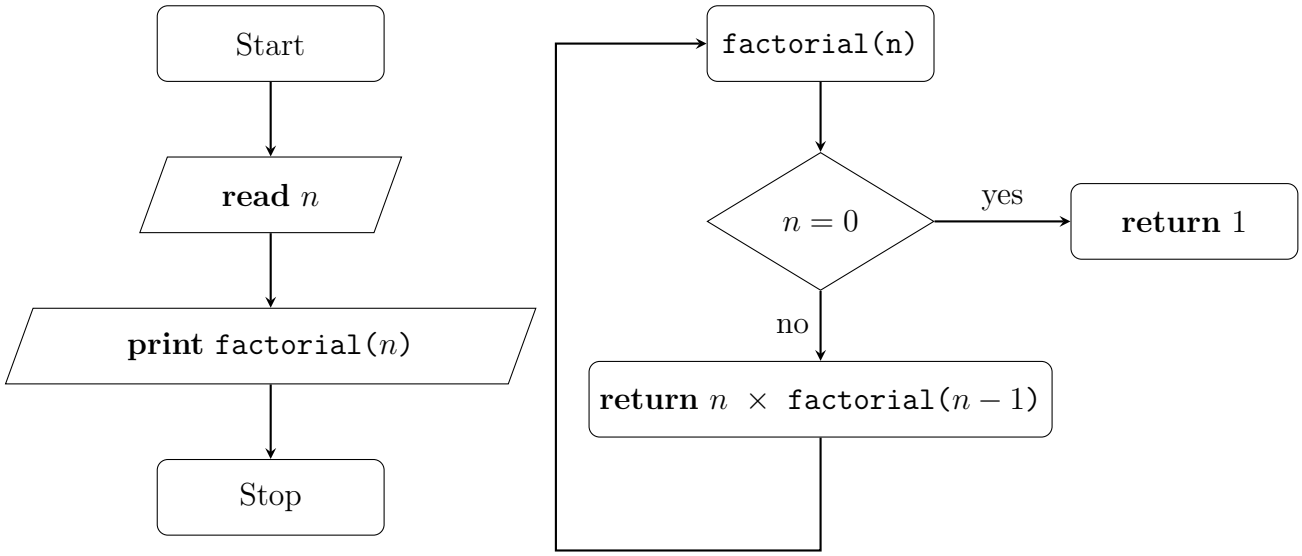
Description:

Program that finds the factorial of a number input by the user by using a recursive function.

Algorithm:

1: Start
2: **function** FACTORIAL(n)
3: **if** $n = 0$ **then**
4: **return** 1
5: **else**
6: **return** $n \times$ FACTORIAL$(n-1)$
7: **read** n
8: **print** FACTORIAL(n)
9: Stop

Flowchart:



Program:

```c
#include <stdio.h>

// Recursive implementation of the factorial function,
// similar to how it's mathematically stated in terms of itself.
int factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n - 1);
}

int main() {
    int n;

    // Read n
    printf("Enter number: ");
    scanf("%d", &n);

    // Output factorial
    printf("Factorial of %d is %d", n, factorial(n));

    return 0;
}
```

Input & Output:

```
Enter number: 5
Factorial of 5 is 120
```

Experiment–6

Aim:

Write a C program to find GCD of two integers by using a recursive function.

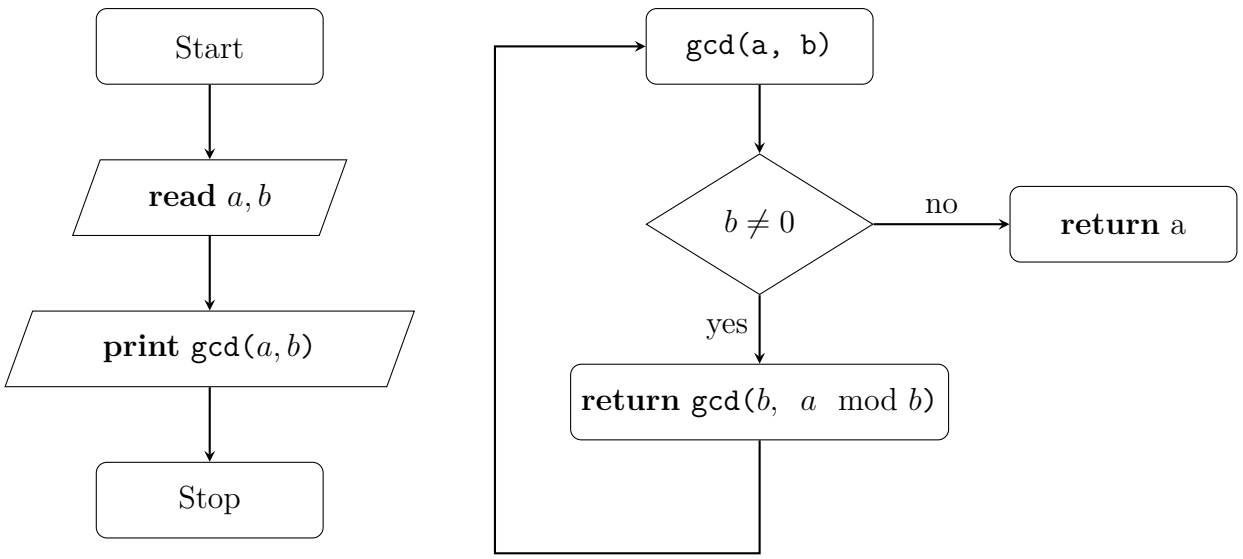
Description:

Program that finds the greatest common denominator of two integers input by the user using a recursive function.

Algorithm:

1: Start
2: **function** GCD(a, b)
3: | **if** $b \neq 0$ **then**
4: | | **return** GCD(b, $a \mod b$)
5: | **else**
6: | | **return** a
7: **read** a, b
8: **print** GCD(a, b)
9: Stop

Flowchart:



Program:

```c
#include <stdio.h>

// Recursive function that calculates the greatest common denominator for 2 positive integers.
int gcd(int a, int b) {
    if (b != 0) {
        return gcd(b, a % b);
    }
    return a;
}

int main() {
    int a, b;

    // Read a
    printf("Enter first number: ");
    scanf("%d", &a);

    // Read b
    printf("Enter second number: ");
    scanf("%d", &b);

    // Output the greatest common denominator of a and b
    printf("GCD(%d, %d) = %d", a, b, gcd(a, b));

    return 0;
}
```

Input & Output:

```
Enter first number: 5
Enter second number: 10
GCD(5, 10) = 5
```

Experiment–7

Aim:

Write a C program to find the largest and smallest number in a list of integers.

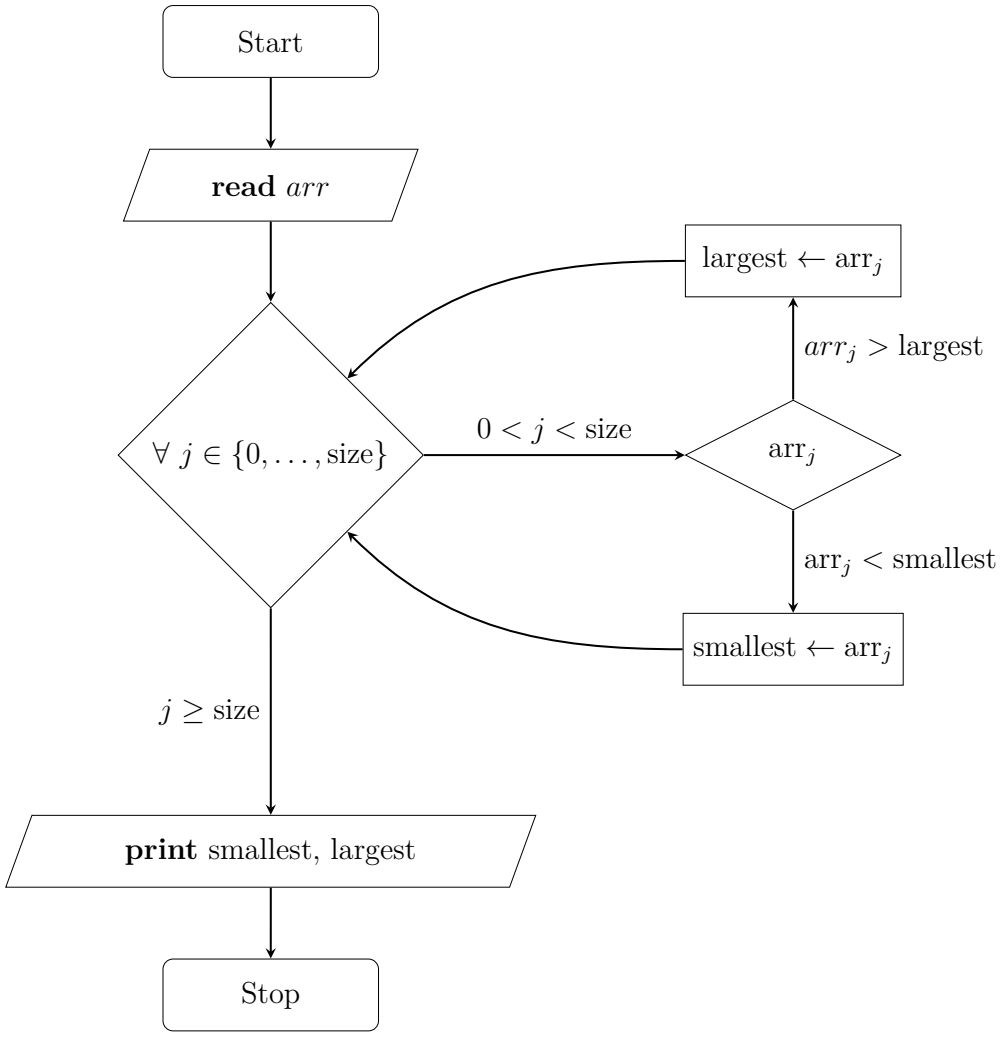
Description:

Program to find the largest and smallest number in an array

Algorithm:

1: Start
2: **read** arr
3: smallest, largest ← arr[0]
4: smallest
5: **for** $\forall\ j \in \{0, \ldots, \text{size}\}$ **do**
6: ⎸ **if** $arr_j <$ smallest **then**
7: ⎸ ⎸ smallest ← arr_j
8: ⎸ **else if** $\text{arr}_j >$ largest **then**
9: ⎸ ⎸ largest ← arr_j
10: **print** smallest, largest
11: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    int size = 5, arr[size];

    // Read array
    for (int i = 0; i < size; i++) {
        // Another approach would be to merge the second loop right here
        printf("Enter number %i: ", i+1);
        scanf("%d", &arr[i]);
    }

    int smallest, largest;
    smallest = largest = arr[0];

    // Find largest and smallest
    for (int j = 0; j < size; j++) {
        if (arr[j] < smallest) {
            smallest = arr[j];
        }

        if (arr[j] > largest) {
            largest = arr[j];
        }
    }

    // Output largest and smallest
    printf("Smallest: %d\nLargest: %d", smallest, largest);

    return 0;
}
```

Input & Output:

```
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
Enter number 4: 4
Enter number 5: 5
Smallest: 1
Largest: 5
```

Experiment–8

Aim:

Write a C program to find the largest and smallest number in a list of integers.

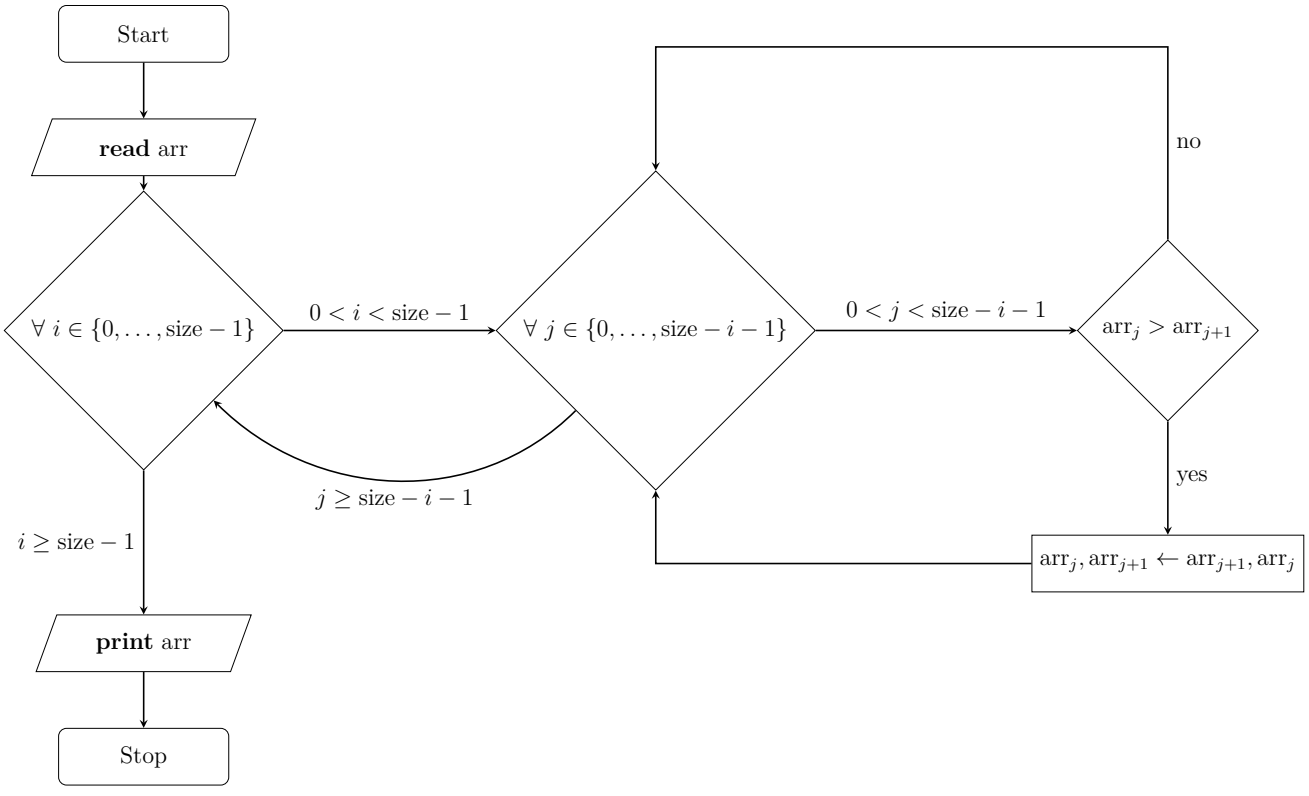
Description:

Program that sorts an array input by the user in ascending order by using bubble sort.

Algorithm:

1: Start
2: **read** arr
3: **for** $\forall\ i \in \{0, \ldots, \text{size} - 1\}$ **do**
4: $\quad$ **for** $\forall\ j \in \{0, \ldots, \text{size} - i - 1\}$ **do**
5: $\quad\quad$ **if** $\text{arr}_j > \text{arr}_{j+1}$ **then**
6: $\quad\quad\quad$ $\text{arr}_j, \text{arr}_{j+1} \leftarrow \text{arr}_{j+1}, \text{arr}_j$
7: **print** arr
8: Stop

Flowchart:



Program:

```c
#include <stdio.h>

int main() {
    int size = 5, arr[size], temp;

    // Read array
    for (int i = 0; i < size; i++) {
        printf("Enter number %i: ", i+1);
        scanf("%d", &arr[i]);
    }

    // Sort the array with bubble sort
    for (int i = 0; i < (size - 1); i++) {
        // Since the last element after each iteration completed in the first loop ensures the greatest number bubbles
        // up to the highest index the second loop need not traverse the whole array.
        for (int j = 0; j < (size - i - 1); j++) {
            // Swap the current and the next number if they're out of order.
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    // Output the sorted array
    printf("[");
    for (int k = 0; k < (size - 1); k++) {
        printf(" '%d' ", arr[k]);
    }
    if (size > 0) {
        printf(" '%d' ", arr[size-1]);
    }
    printf("]");

    return 0;
}
```

Input & Output:

```
Enter number 1: 5
Enter number 2: 4
Enter number 3: 3
Enter number 4: 2
Enter number 5: 1
[ '1'  '2'  '3'  '4'  '5' ]
```

Experiment–9

Aim:

Write a C program to multiply two matrices.

Description:

If A is an $m \times n$ matrix and B is an $n \times p$ matrix, such that:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

Then the matrix product $C = AB$ is defined to be a $m \times p$ matrix:

$$C = AB = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

Where,

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}, \text{ for } i = 1, \ \ldots, \ m \text{ and } j = 1, \ \ldots, \ p$$

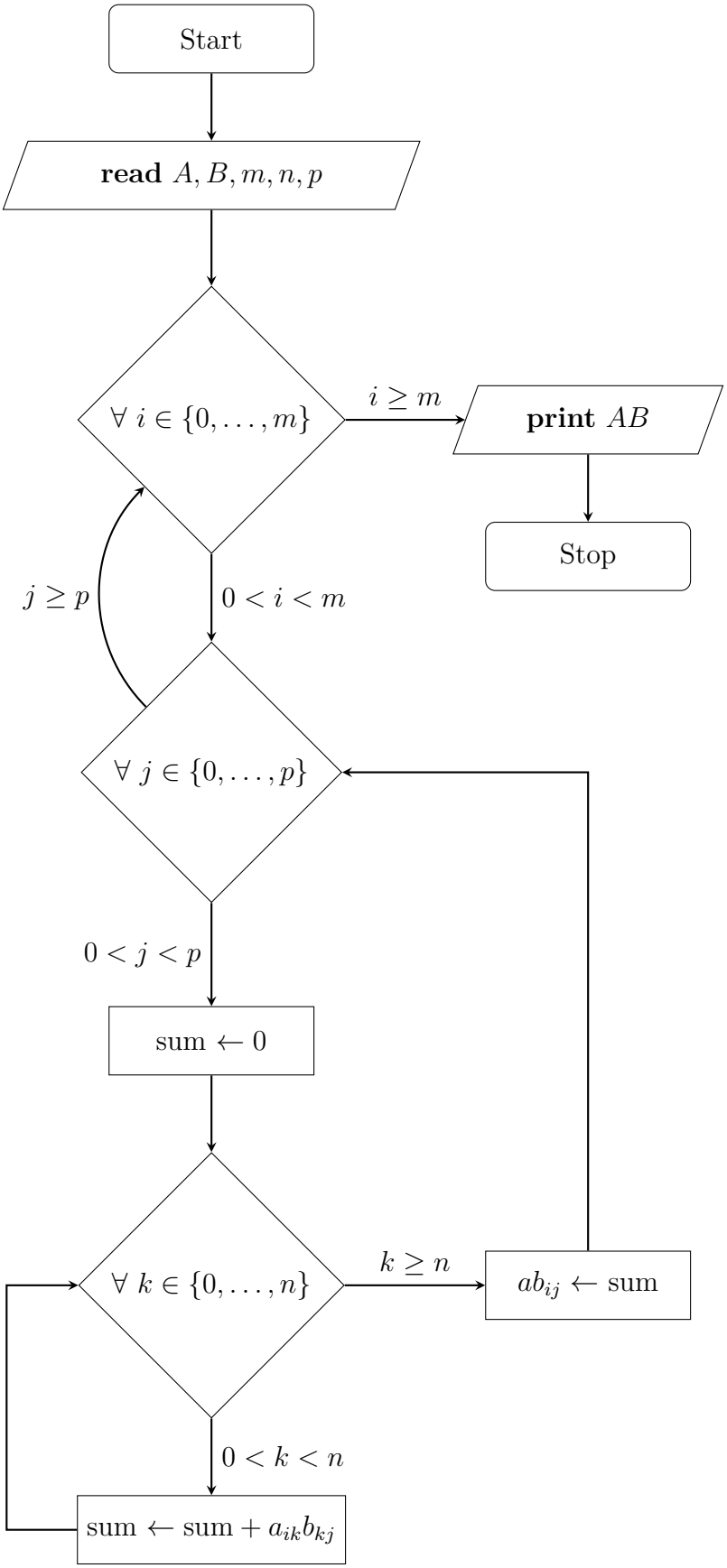
Example:

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} (0 \times 0 + 1 \times 2 + 2 \times 4) & (0 \times 1 + 1 \times 3 + 2 \times 5) \\ (3 \times 0 + 4 \times 2 + 5 \times 4) & (3 \times 1 + 4 \times 3 + 5 \times 5) \end{bmatrix} = \begin{bmatrix} 10 & 13 \\ 28 & 40 \end{bmatrix}$$

Algorithm:

1: Start
2: **read** A, B, m, n, p
3: **for** $\forall\ i \in \{0, \ldots, m\}$ **do**
4: ⎸ **for** $\forall\ j \in \{0, \ldots, p\}$ **do**
5: ⎸ ⎸ sum $\leftarrow 0$;
6: ⎸ ⎸ **for** $\forall\ k \in \{0, \ldots, n\}$ **do**
7: ⎸ ⎸ ⎸ sum $\leftarrow$ sum $+ a_{ik}b_{kj}$
8: ⎸ ⎸ $ab_{ij} \leftarrow$ sum
9: **print** AB
10: Stop

Flowchart:


Start
read A, B, m, n, p
$\forall\ i \in \{0, \ldots, m\}$
$i \geq m$
print AB
Stop
$j \geq p$
$0 < i < m$
$\forall\ j \in \{0, \ldots, p\}$
$0 < j < p$
sum $\leftarrow 0$
$\forall\ k \in \{0, \ldots, n\}$
$k \geq n$
$ab_{ij} \leftarrow$ sum
$0 < k < n$
sum $\leftarrow$ sum $+ a_{ik}b_{kj}$

Program:

```c
#include <stdio.h>

void inputMatrix(char name, int arr[][10], int rows, int columns) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            printf("Enter %c_%d%d: ", name, i + 1, j + 1);
            scanf("%d", &(arr[i][j]));
        }
    }
}

void printMatrix(int arr[][10], int rows, int columns) {
    for (int i = 0; i < rows; i++) {
        printf("| ");
        for (int j = 0; j < columns; j++) {
            printf("%d ", arr[i][j]);
        }
    }
}

void main() {
    int m, n, p, A[10][10], B[10][10], AB[10][10], sum;

    // Read number of rows and columns for first matrix
    printf("Enter number of rows x columns for matrix A (<10): ");
    scanf("%dx%d", &m, &n);

    // The number of rows must be equal to the number of columns in the
    ↪ first matrix
    printf("Enter number of columns for matrix B (<10): ");
    scanf("%d", &p);

    // Read values for both matrices
    inputMatrix('A', A, m, n); matrixReader('B', B, n, p);

    // Calculate matrix multiplication [O(n^3)]
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            sum = 0;
            for (int k = 0; k < n; k++) {sum += A[i][k] * B[k][j];}
            AB[i][j] = sum;
        }
    }

    // Print matrix multiplication
    printf("\nThe matrix product AB =");
    matrixPrinter(AB, m, p);
}
```

Input & Output:

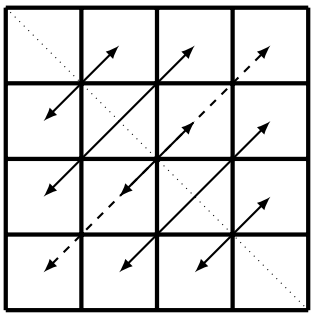
```
Enter number of rows for matrix A (<10): 2x3
Enter number of columns for matrix B (<10): 2

Enter A_11: 0
Enter A_12: 1
Enter A_13: 2
Enter A_21: 3
Enter A_22: 4
Enter A_23: 5

Enter B_11: 0
Enter B_12: 1
Enter B_21: 2
Enter B_22: 3
Enter B_31: 4
Enter B_32: 5

The matrix product AB =
| 10 13 |
| 28 40 |
```

Experiment–10

Aim:

Write a C program to check whether a matrix is symmetric or not.

Description:



A symmetric matrix is a square matrix that is equal to its transpose. Formally described as:

$$\boxed{\text{A is symmetric} \iff A = A^T}$$

Or,

$$\boxed{\text{A is symmetric} \iff \forall\ i, j, \quad a_{ji} = a_{ij}}$$

For example, the following 3×3 matrix is symmetric:

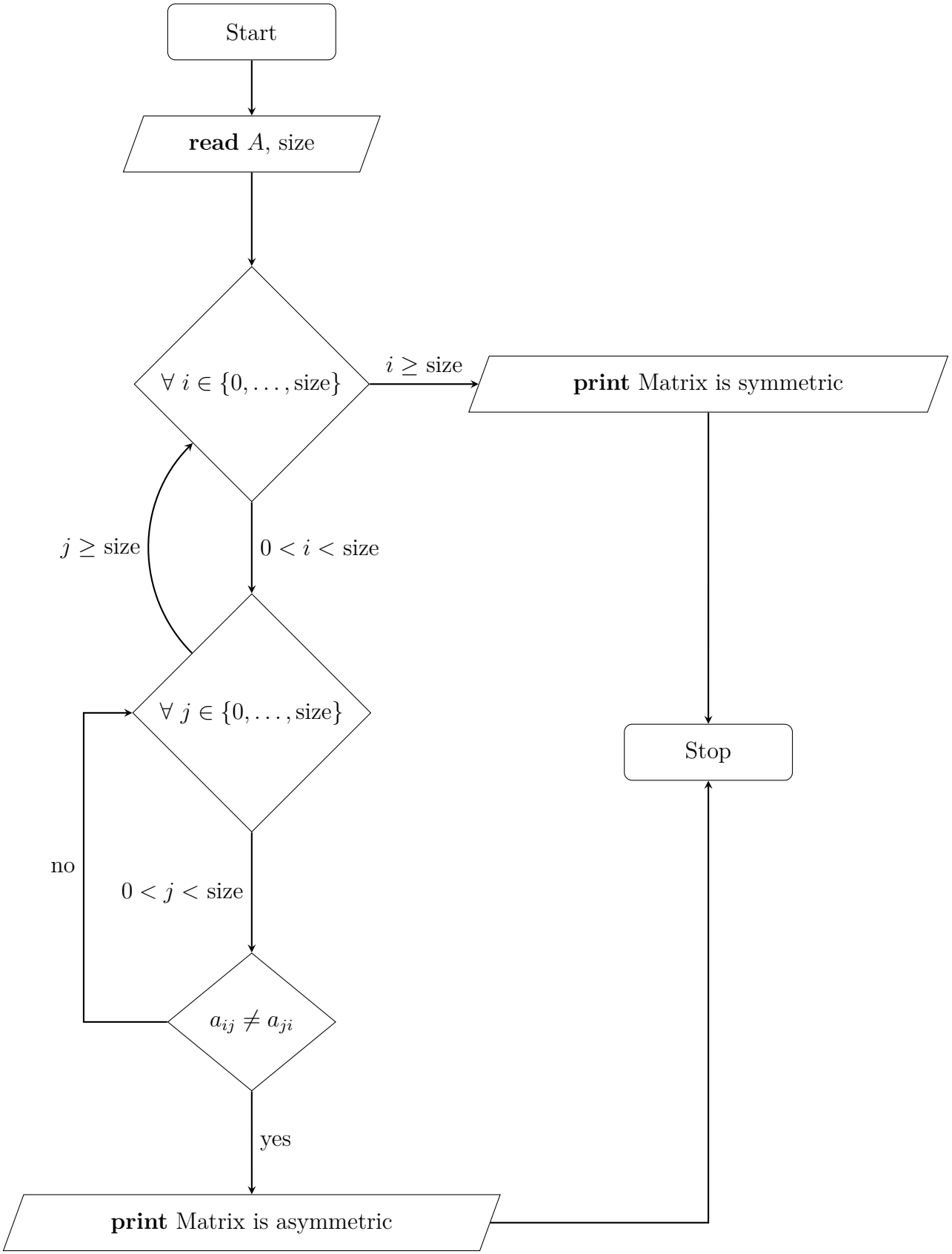
$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Since $A = A^T$

Algorithm:

```
1: Start
2: read A, size
3: for ∀ i ∈ {0, ..., size} do
4:     for ∀ j ∈ {0, ..., size} do
5:         if a_ij ≠ a_ji then
6:             print Matrix isn't symmetric
7:             goto Stop
8: print Matrix is symmetric
9: Stop
```

Flowchart:


Start
read A, size
$\forall\, i \in \{0, \ldots, \text{size}\}$
$i \geq$ size
print Matrix is symmetric
$j \geq$ size
$0 < i <$ size
$\forall\, j \in \{0, \ldots, \text{size}\}$
Stop
no
$0 < j <$ size
$a_{ij} \neq a_{ji}$
yes
print Matrix is asymmetric

Program:

```c
#include <stdio.h>

void inputMatrix(char name, int arr[][10], int rows, int columns) {
    printf("\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            printf("Enter %c_%d%d: ", name, i + 1, j + 1);
            scanf("%d", &(arr[i][j]));
        }
    }
}
void printMatrix(int arr[][10], int rows, int columns) {
    printf("\n");
    for (int i = 0; i < rows; i++) {
        printf("| ");
        for (int j = 0; j < columns; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("|\n");
    }
}
int main() {
    int size, matrix[10][10];

    // Read number of rows and columns
    printf("Enter size of square matrix (<10): ");
    scanf("%d", &size);
    // Read values for both matrices
    inputMatrix('a', matrix, size, size);

    // Check symmetricality
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++) {
            if (matrix[i][j] != matrix[j][i]) {
                // Asymmetric elements found, abort.
                printf("\nThe given given matrix, A =");
                printMatrix(matrix, size, size);
                printf("is not symmetric as a_%d%d does not equal a_%d%d",
                ↪  i + 1, j + 1, j + 1, i + 1);
                return 0;
            }
        }
    }

    // This step will only be reached if no asymmetric elements are found
    printf("\nThe given given matrix, A =");
    printMatrix(matrix, size, size);
    printf("is symmetric.");
    return 0;
}
```

Input & Output (Symmetric):

```
Enter size of square matrix (<10): 3

Enter a_11: 1
Enter a_12: 2
Enter a_13: 3
Enter a_21: 2
Enter a_22: 6
Enter a_23: 4
Enter a_31: 3
Enter a_32: 4
Enter a_33: 5

The given given matrix, A =
| 1 2 3 |
| 2 6 4 |
| 3 4 5 |
is symmetric.
```

Input & Output (Asymmetric):

```
Enter size of square matrix (<10): 3

Enter a_11: 1
Enter a_12: 2
Enter a_13: 3
Enter a_21: 4
Enter a_22: 5
Enter a_23: 6
Enter a_31: 7
Enter a_32: 8
Enter a_33: 9

The given given matrix, A =
| 1 2 3 |
| 4 5 6 |
| 7 8 9 |
is not symmetric as a_12 does not equal a_21
```