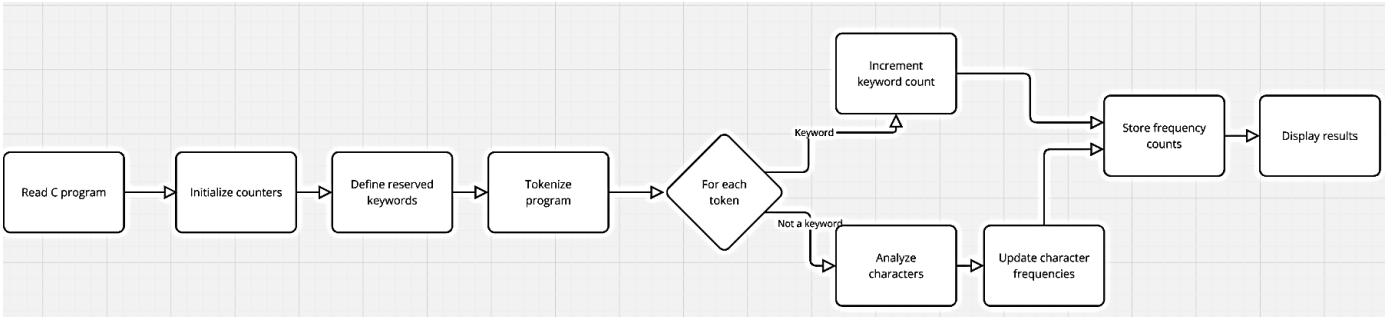
Draw the detailed flowchart of the following problems. Mention your assumptions. Make a sufficient number of dry runs to test your algorithm.

a. Frequency Analysis of Characters and Reserved Words in a C Program
Flow chart



Algorithm

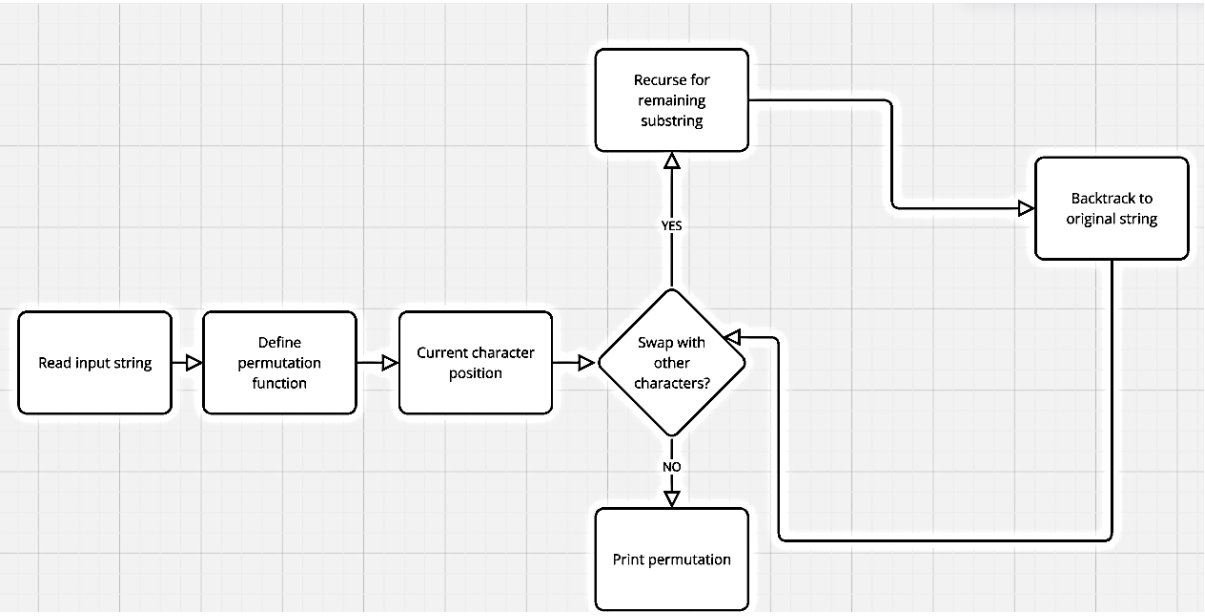
Start:
Step 1: Read the input C program as a string or line by line.
Step 2: Initialize counters for alphabets, digits, special characters, and whitespace.
Step 3: Define a list of reserved keywords in C.
Step 4: Tokenize the program into words.
Step 5: For each token:

- Check if it matches any reserved keyword.
- If it is a reserved keyword, increment its count.
- If not, analyze its characters and update their frequencies.

Step 6: Store the frequency counts for each character and reserved word.
Step 7: Display the frequency results.
End

b. Generate All Permutations of n Characters

Flow chart:



Algorithm

Start:

Step 1: Read the input string of n characters.

Step 2: Define a function for generating permutations recursively.

Step 3: For the current character position:

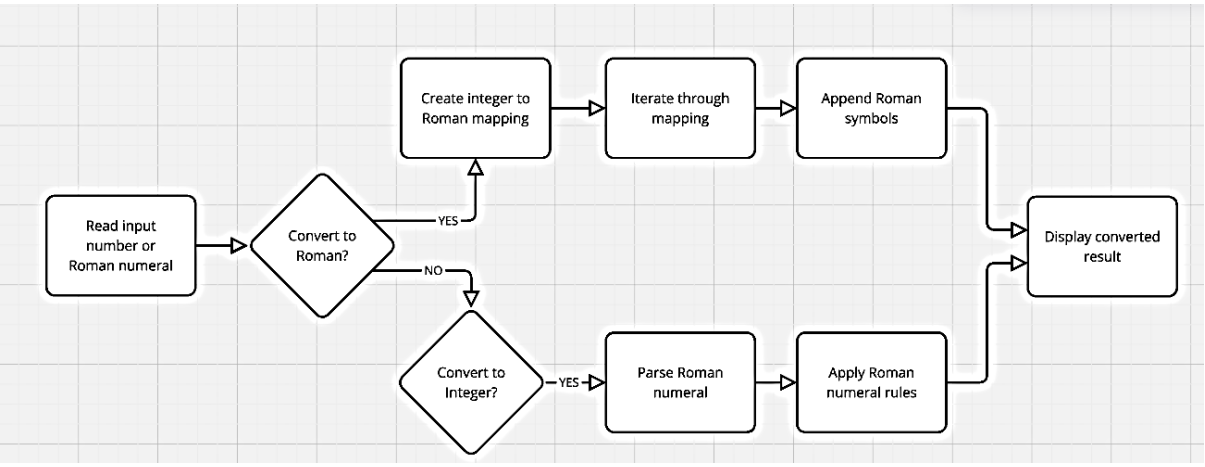
- Swap the current character with every other character.
- Recurse to generate permutations for the remaining substring.
- Backtrack by swapping back to restore the original string.

Step 4: Print each generated permutation.

End

c. Convert Integer to Roman and Vice Versa

Flowchart :



Algorithm :

Start:

Step 1: Read the input number or Roman numeral.

Step 2: If converting integer to Roman:

- Create a mapping of integers to Roman numeral symbols.
- Iterate through the mapping in descending order.
- Append Roman symbols while subtracting their values from the input number.

Step 3: If converting Roman to integer:

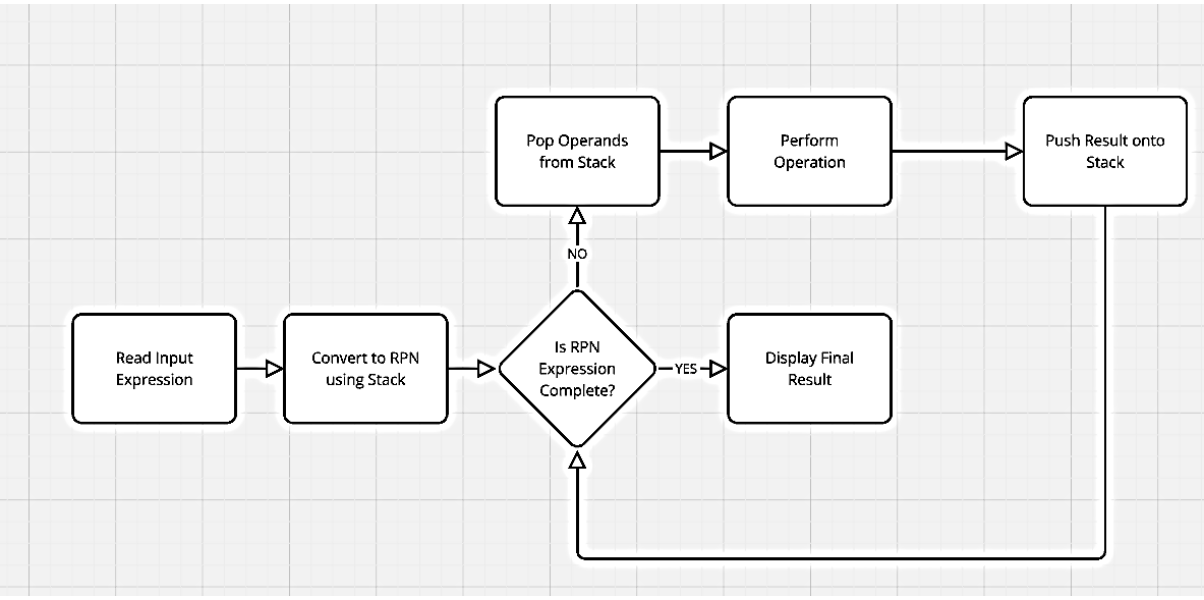
- Parse the Roman numeral from left to right.
- Add or subtract values based on Roman numeral rules.s

Step 4: Display the converted result.

End

d. Evaluate Arithmetic Expression

Flowchart :



Algorithm :

Start:

Step 1: Read the input arithmetic expression.

Step 2: Convert the expression into Reverse Polish Notation (RPN) using a stack.

Step 3: Evaluate the RPN expression:

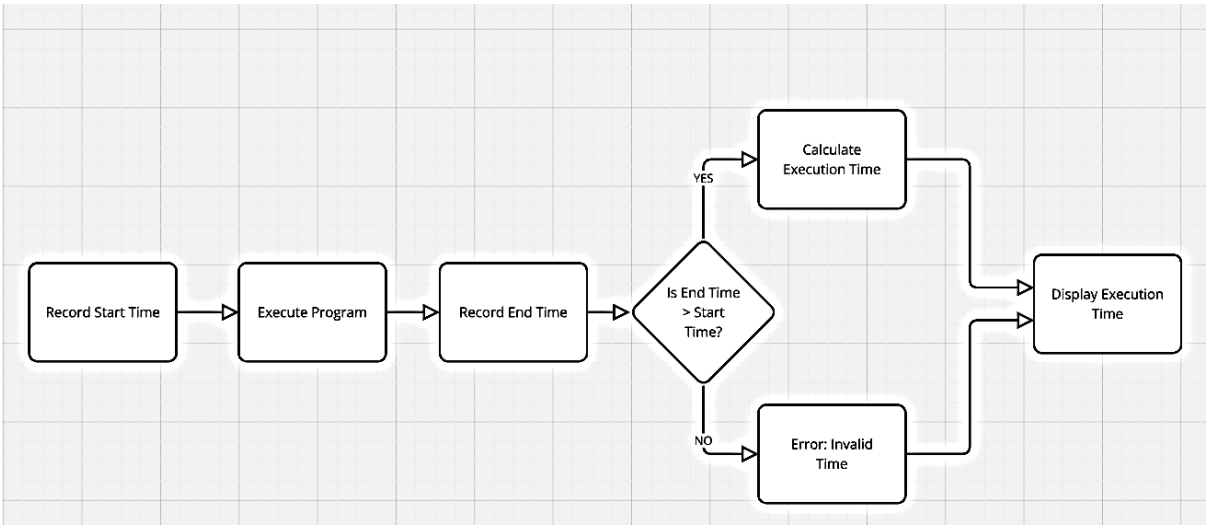
- For each operator, pop operands from the stack.
- Perform the operation and push the result back onto the stack.

Step 4: Display the final result from the stack.

End

e. Find Execution Time of a Program Without Timestamps

Flowchart :



Algorithm :

Start:

Step 1: Use a function like clock() to record the program's start time.

Step 2: Execute the program or function to be measured.

Step 3: Record the program's end time using clock().

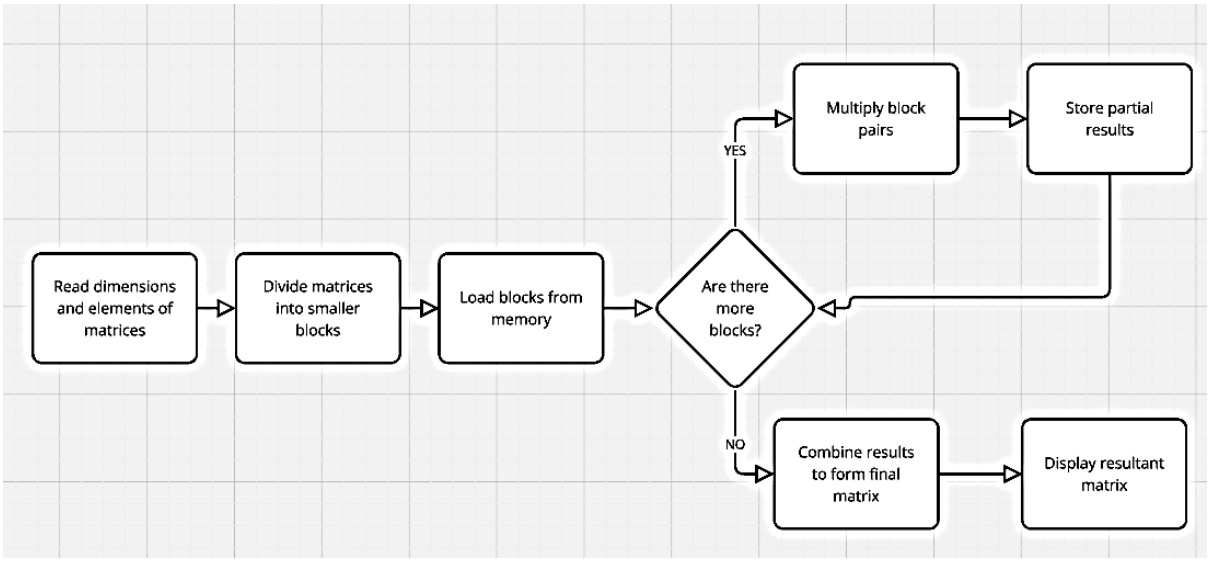
Step 4: Calculate the execution time as (end_time - start_time) / CLOCKS_PER_SEC.

Step 5: Display the execution time.

End

f. Matrix Multiplication of Large Matrices

Flowchart :



Algorithm :

Start:

Step 1: Read the dimensions and elements of two matrices.

Step 2: Divide the matrices into smaller blocks that fit into memory.

Step 3: Multiply corresponding blocks of matrices:

- Load blocks from memory.
- Perform multiplication for each block pair.
- Store partial results in a temporary matrix.

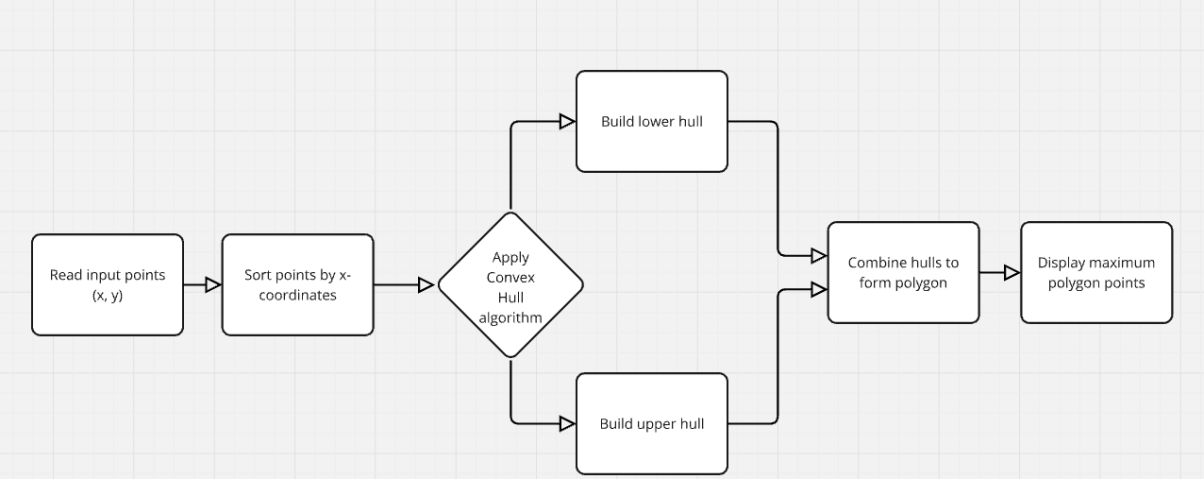
Step 4: Combine the results to form the final matrix.

Step 5: Display the resultant matrix.

End

g. Find Maximum Polygon from Points

Flowchart :



Algorithm :

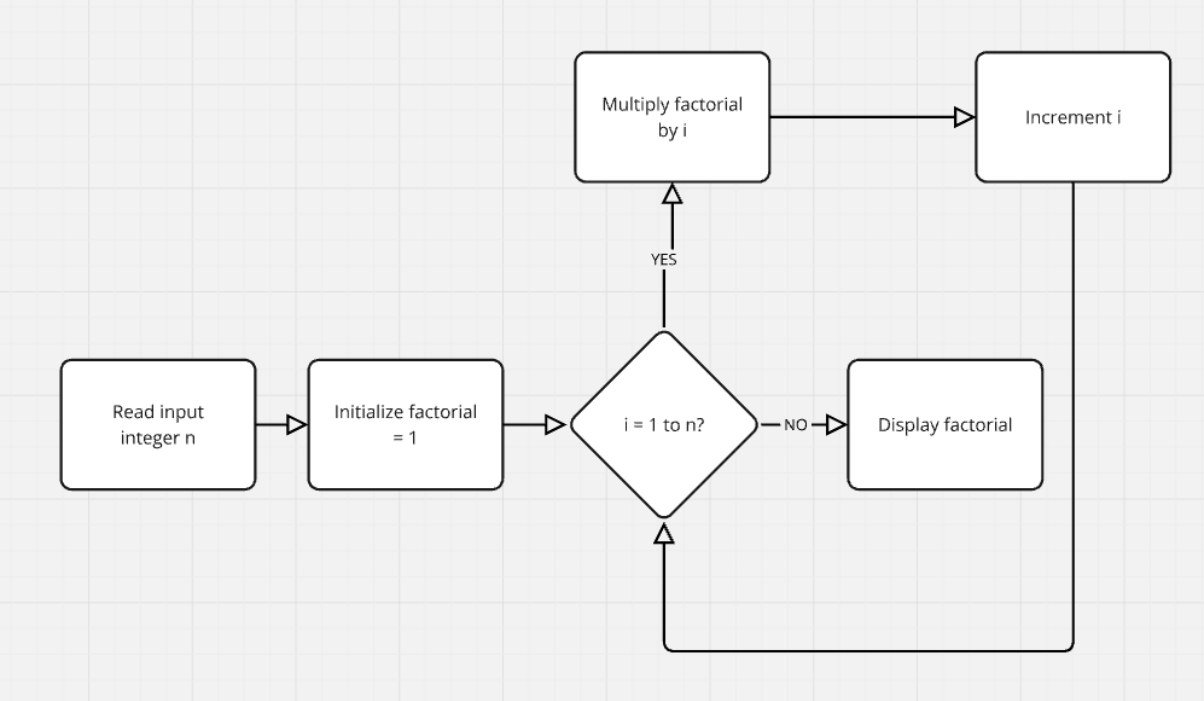
Start:
Step 1: Read the input set of points (x, y).
Step 2: Sort the points based on x-coordinates.
Step 3: Apply the Convex Hull algorithm:

- Build the lower hull by iterating over the sorted points.
- Build the upper hull in reverse order.

Step 4: Combine the upper and lower hulls to form the convex polygon.
Step 5: Display the points that form the maximum polygon.
End

h. Factorial of an Integer (Up to 10-Digit Numbers)

Flowchart :



Algorithm :

Start:
Step 1: Read the input integer n.
Step 2: Initialize a variable factorial = 1.
Step 3: For i = 1 to n:

- Multiply factorial by i.

Step 4: Display the value of factorial.
End