4. Take a sufficiently long expression with +,-,\*,/ operators including paranthesis. Evaluate the expression.

**Constraints:** Numbers in the infix expression should be of single digit.

**Input Example**

Enter a valid infix expression

6+7-(5\*3-7)+2

**Output Example**

The result of this expression is 7.0000

**Algorithm**

Step 1: Start

Step 2: Take input string as the infix expression which has to be calculated.

Step 3: Take an global empty stack initially.

Step 4: Take an empty string postfix.

Step 5: Take a double result and set result🡨0.

Step 6: Call the function infix\_To\_postfix(str) and set postfix🡨 infix\_To\_postfix(str).

Step 7: Call the function postfix\_To\_result(postfix) and result🡨 postfix\_To\_result(postfix).

Step 8: Print the result.

Step 9: End.

**Step 6:** Algorithm of the function in step6 is as follows:

string infix\_To\_postfix(string str)

Step 6.1: Set i🡨0 and iterate over character by character to scan the infix expression from left to right.

Step 6.2.1: If the scanned character is an operand, output it.

Step 6.2.2: Else, check if the precedence and associativity of the scanned operator is greater than the precedence and associativity of the operator in top of the stack(or the stack is empty or the stack contains a ‘(‘ ), then push it to the stack.

Step 6.3.1: Operators ‘+’,’-‘,’\*’ and ‘/’ are left-associative. When the top stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.

Step 6.3.2: Else,pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that push the scanned operator to the stack. If ‘)’ paranthesis is encountered while popping then stop and push the scanned operator in the stack.

Step 6.4: If the scanned character is an ‘(‘, push it to the stack. If the scanned character is an ‘)’, pop the stack and output it until a ‘(‘ is encountered.

Step 6.5: Repeat steps 6.2-6.4 until i!= strlen(str).

Step 6.6: Pop and output from the stack until it is not empty.

Step 6.7: End of the function,return the string postfix to the main program.

**Step 7:** Algorithm of the function in step7 is as follows:

double postfix\_To\_result(postfix)

Step 7.1: Set i🡨0 and iterate over character by character to scan the string postfix taken as argument from left to right.

Step 7.2.1: If the character is an oparand, then push it into the stack

Step 7.2.2: Else,if the character is an operator, pop operands for that operator from the stack. Evaluate the operator and push the result back to the stack

Step 7.3: Repeat step 7.2 until i!= strlen(str).

Step 7.4: After the end of the iteration, the number in the top of the stack is the answer. Return the number from the function as result to the main program.