Q1: Write an algorithm to check if an expression has balanced parenthesis using stack. [3]

Ans:

Procedure check()

- Declare a character stack S.
- Now traverse the expression.
  a) If the current character is a starting bracket then push it to stack.
  b) If the current character is a closing bracket then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.

- After complete traversal, if there is some starting bracket left in stack then “not balanced”.

End procedure

EXAMPLE:

Eg: [a+(b*c)+{(d-e)}]

Thus, parenthesis match here
Q2: Write an algorithm to convert infix expression into postfix expression with parenthesis. [07]

Ans:

The algorithm uses a stack to temporarily hold operators and left parentheses. The postfix expression P will be constructed from left to right using the operands from Q and the operators which are removed from STACK. We begin by pushing a left parenthesis onto STACK and adding right parenthesis at the end of Q. The algorithm is completed when STACK is empty.

Algorithm 6.6: POLISH(Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P:

1. Push “(” onto STACK, and add “)” to the end of Q.
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator ⊗ is encountered, then:

   (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than ⊗.

   (b) Add ⊗ to STACK.
    [End of If structure.]

6. If a right parenthesis is encountered, then:

   (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.

   (b) Remove the left parenthesis. [Do not add the left parenthesis to P.]
    [End of If structure.]
    [End of Step 2 loop.]

7. Exit.
Q3: Define recursion. What care should be taken in writing recursive function?

Ans:

Recursion is an important concept in computer science. Many algorithms can be best described in terms of recursion.

Suppose P is a procedure containing either a Call statement to itself or a Call statement to a second procedure that may eventually result in a Call statement back to the original procedure P. Then P is called a recursive procedure. So that the program will not continue to run indefinitely, a recursive procedure must have the following two properties:

1. There must be certain criteria, called base criteria, for which the procedure does not call itself.
2. Each time the procedure does call itself (directly or indirectly), it must be closer to the base criteria.

Q4: Write a recursive algorithm to find factorial.

Ans: Note: here I have mention both algo. (with and without recursion for your understanding only)

Without Recursion

**Procedure 6.9A:** FACTORIAL(FACT, N)

This procedure calculates N! and returns the value in the variable FACT.

1. If N = 0, then: Set FACT := 1, and Return.
2. Set FACT := 1. [Initializes FACT for loop.]
3. Repeat for K = 1 to N.
   - Set FACT := K*FACT.
   [End of loop.]
4. Return.

With Recursion

**Procedure 6.9B:** FACTORIAL(FACT, N)

This procedure calculates N! and returns the value in the variable FACT.

1. If N = 0, then: Set FACT := 1, and Return.
2. Call FACTORIAL(FUNCT, N - 1).
3. Set FACT := N*FACT.
4. Return.
Q5: Define recursion. What care should be taken in writing recursive function? Give a recursive solution for the problem of “Towers of Hanoi”. [7]

Ans:

(Note: First question’s ans included in ans of question 3)

**Towers of Hanoi:**

Problem:

Suppose three pegs, labeled A, B and C, are given, and suppose on peg A there are placed a finite number n of disks with decreasing size. This is pictured in Fig. 6.14 for the case n = 6. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary. The rules of the game are as follows:

(a) Only one disk may be moved at a time. Specifically, only the top disk on any peg may be moved to any other peg.
(b) At no time can a larger disk be placed on a smaller disk.

![Fig. 6.14 Initial Setup of Towers of Hanoi with n = 6](image)
Example : n=3

Recursive Solution:
Q6: What is the advantage of postfix expression over infix expression? Write an algorithm of postfix expression evaluation. [7]

Ans:

**Advantages of postfix expression:**

- It is very easily implemented and does not have overhead of parenthesis.
- It is good for machines because they can be parsed and evaluated easily.

**Algorithm for Evaluation of Postfix Expression:**

```
Evaluation of a Postfix Expression

Suppose P is an arithmetic expression written in postfix notation. The following algorithm, which uses a STACK to hold operands, evaluates P.

Algorithm 6.5: This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.
1. Add a right parenthesis “)" at the end of P. [This acts as a sentinel.]
2. Scan P from left to right and repeat Steps 3 and 4 for each element of P until the sentinel “)” is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator ⊗ is encountered, then:
   (a) Remove the two top elements of STACK, where A is the top element and B is the next to top element.
   (b) Evaluate B ⊗ A.
   (c) Place the result of (b) back on STACK.
   [End of If structure.]
   [End of Step 2 loop.]
5. Set VALUE equal to the top element on STACK.

We note that, when Step 5 is executed, there should be only one number on STACK.
```
Q7: evaluate the following expression showing every status of stack in tabular form. [7]

**Evaluate (i):** \( 546 + 493 / + * \)

1. **Empty Stack**

2. **Read and push operands 5, 4, 6**
   - Stack:
     - 5
     - 4
     - 6

3. **Read Operator +, pop two values from stack opn2 = 6, opn1 = 4, and push the answer 10**
   - Stack:
     - 10
     - 5

4. **Read Operator *, pop two values from stack opn2 = 10, opn1 = 5, and push the answer 50**
   - Stack:
     - 50
     - 350

   **Popped value 350 is the answer**

5. **Read Operator *, pop two values from stack opn2 = 7, opn1 = 50, and push the answer 350**
   - Stack:
     - 350
     - 350

6. **Read Operator /, pop two values from stack opn2 = 3, opn1 = 4, and push the answer 7**
   - Stack:
     - 7
     - 4

7. **Read Operator /, pop two values from stack opn2 = 3, opn1 = 9, and push the answer 3**
   - Stack:
     - 3
     - 9

8. **Read and push operands 4, 9, 3**
   - Stack:
     - 3
     - 9
     - 4

---

**Evaluate (ii):** \( 752 + 411 + / - \)

1. **Empty Stack**

2. **Read and push operands 7, 5, 2**
   - Stack:
     - 2
     - 5
     - 7

3. **Read Operator +, pop two values from stack opn2 = 2, opn1 = 5, and push the answer 7**
   - Stack:
     - 7
     - 7

4. **Read Operator *, pop two values from stack opn2 = 7, opn1 = 7, and push the answer 49**
   - Stack:
     - 49
     - 47

5. **Read Operator -, pop two values from stack opn2 = 2, opn1 = 49, and push the answer 47**
   - Stack:
     - 47
     - 49

6. **Read Operator /, pop two values from stack opn2 = 2, opn1 = 4, and push the answer 2**
   - Stack:
     - 2
     - 4

7. **Read Operator +, pop two values from stack opn2 = 1, opn1 = 1, and push the answer 2**
   - Stack:
     - 2
     - 1

   **Popped value 47 is the answer**

---

Compiled By: Kaushik Vaghani
Subject Faculties: Yatin Shukla, Kaushik Vaghani, Varsha Naregalkar, Kruti Sheth
Q8: Applications of Stack [3]

Ans:

- Recursion handling
- Evaluation of expression
  - Conversion of infix to postfix expression
  - Computation of postfix expression
- Parenthesis handling
- Backtracking
  - Conversion of decimal to other number system
  - Undo operations