Evaluation of Postfix

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis is not required in postfix. We have discussed infix to postfix conversion. In this post, the evaluation of postfix expressions is discussed.

Example:

> **Input**: str = "2 3 1 * + 9 -"
> **Output**: -4
> ***Explanation:***
>
> - Scan 2, it's a number, so push it to stack. Stack contains '2'
> - Scan 3, again a number, push it to stack, stack now contains '2 3' (from bottom to top)
> - Scan 1, again a number, push it to stack, stack now contains '2 3 1'
> - Scan *, it's an operator, pop two operands from stack, apply the * operator on operands, we get 3*1 which results in 3. We push the result 3 to stack. The stack now becomes '2 3'.
> - Scan +, it's an operator, pop two operands from stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result 5 to stack. The stack now becomes '5'.
> - Scan 9, it's a number, so we push it to the stack. The stack now becomes '5 9'.
> - Scan -, it's an operator, pop two operands from stack, apply the - operator on operands, we get 5 - 9 which results in -4. We push the result -4 to the stack. The stack now becomes '-4'.

Dash
All
Articles
Videos
Problems
Quiz

> - There are no more elements to scan, we return the top element from the stack (which is the only element left in a stack).
>
> **Input**: str = "100 200 + 2 / 5 * 7 +"
> **Output**: 757

Evaluation of Postfix Expression Using Stack:

Follow the steps mentioned below to evaluate postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
  - If the element is a number, push it into the stack
  - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack
- When the expression is ended, the number in the stack is the final answer

Below is the implementation of the above approach:

C++ Java

```
// Java program to evaluate value of a postfix expression

import java.util.Stack;

public class Test
{
```

Dash
All
Articles
Videos
Problems
Quiz

```
// Method to evaluate value of a postfix expression
static int evaluatePostfix(String exp)
{
    //create a stack
    Stack<Integer> stack=new Stack<>();

    // Scan all characters one by one
    for(int i=0;i<exp.length();i++)
    {
        char c=exp.charAt(i);

        // If the scanned character is an operand (number here),
        // push it to the stack.
        if(Character.isDigit(c))
        stack.push(c - '0');

        //  If the scanned character is an operator, pop two
        // elements from stack apply the operator
        else
        {
            int val1 = stack.pop();
            int val2 = stack.pop();

            switch(c)
            {
                case '+':
                stack.push(val2+val1);
                break;
```

Dash
All
Articles
Videos
Problems
Quiz

```
                        case '-':
                        stack.push(val2- val1);
                        break;

                        case '/':
                        stack.push(val2/val1);
                        break;

                        case '*':
                        stack.push(val2*val1);
                        break;
                  }
                }
            }
            return stack.pop();
        }

        // Driver program to test above functions
        public static void main(String[] args)
        {
            String exp="231*+9-";
            System.out.println("postfix evaluation: "+evaluatePostfix(exp));
        }
    }
```

Output

```
postfix evaluation: -4
```

Time Complexity: O(N)
Auxiliary Space: O(N)

Dash
All
Articles
Videos
Problems
Quiz

There are the following limitations of the above implementation.

- It supports only **4** binary operators **'+', '*', '-'** and **'/'**. It can be extended for more operators by adding more switch cases.
- The allowed operands are only single-digit operands. The program can be extended for multiple digits by adding a separator-like space between all elements **(operators and operands)** of the given expression.

Below given is the extended program which allows operands to have multiple digits.

C++ Java

```
// Java program to evaluate value of a postfix
// expression having multiple digit operands
import java.util.Stack;

class Test1
{
    // Method to evaluate value of a postfix expression
    static int evaluatePostfix(String exp)
    {
        //create a stack
        Stack<Integer> stack = new Stack<>();

        // Scan all characters one by one
        for(int i = 0; i < exp.length(); i++)
        {
```

```
char c = exp.charAt(i);

if(c == ' ')
continue;

// If the scanned character is an operand
// (number here),extract the number
// Push it to the stack.
else if(Character.isDigit(c))
{
    int n = 0;

    //extract the characters and store it in num
    while(Character.isDigit(c))
    {
        n = n*10 + (int)(c-'0');
        i++;
        c = exp.charAt(i);
    }
    i--;

    //push the number in stack
    stack.push(n);
}

// If the scanned character is an operator, pop two
// elements from stack apply the operator
else
{
```

Dash
All

```
                int val1 = stack.pop();
                int val2 = stack.pop();

                switch(c)
                {
                    case '+':
                    stack.push(val2+val1);
                    break;

                    case '-':
```

Videos
Problems
Quiz

```
                    case '/':
                    stack.push(val2/val1);
                    break;

                    case '*':
                    stack.push(val2*val1);
                    break;
                }
            }
        }
        return stack.pop();
    }

    // Driver program to test above functions
    public static void main(String[] args)
    {
        String exp = "100 200 + 2 / 5 * 7 +";
        System.out.println(evaluatePostfix(exp));
    }
}
```

Output

```
757
```

Time Complexity: O(N)
Auxiliary Space: O(N)

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.