Week 7

Aim: Convert infix expression to postfix expression.

Program:

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <stack>
using namespace std;

int precedence(char x){
    if(x=='+' || x=='-'){
        return 1;
    }
    else if(x=='*' || x=='/'){
        return 2;
    }
    else if(x=='^'){
        return 3;
    }
    return -1;
}

int main() {
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    string s;
    cin>>s;
    string ans="";
    stack<char> st;
    for(int i=0;i<s.length();i++){
        if(s[i]>='a' && s[i]<='z'){
            ans.push_back(s[i]);
        }
        else if(s[i]==')'){
            while(!st.empty() && st.top()!='('){
                ans.push_back(st.top());
                st.pop();
            }
            if(!st.empty()){
                st.pop();
            }
        }
        else{
            if(st.empty()){
```

```
            st.push(s[i]);
        }
        else{
            int y=precedence(st.top());
            int x=precedence(s[i]);
            while(y>=x){
                ans.push_back(st.top());
                st.pop();
                if(st.empty()){
                    break;
                }
               y=precedence(st.top());
            }
            st.push(s[i]);
        }
    }
  }
  while(!st.empty()){
    ans.push_back(st.top());
    st.pop();
  }
  cout<<ans<<endl;
  return 0;
}
```

Input & Output:

Testcase 0 ✔ | Testcase 1 ✔ | Testcase 2 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Compile Message

```
Solution.cpp: In function 'int main()':
Solution.cpp:28:18: warning: comparison of integer expressions of different signedness: 'int' and 'std::__cxx11::basic_string<char
>::size_type' {aka 'long unsigned int'} [-Wsign-compare]
     for(int i=0;i<s.length();i++){
                 ~^~~~~~~~~~
```

Compile Time

Input (stdin)

```
(a+b)
```

Run Time

Your Output (stdout)

```
ab+
```

Expected Output

```
ab+
```

Testcase 0 ✔ | Testcase 1 ✔ | Testcase 2 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Compile Time

Compile Message

```
Solution.cpp: In function 'int main()':
Solution.cpp:28:18: warning: comparison of integer expressions of different signedness: 'int' and 'std::__cxx11::basic_string<char
>::size_type' {aka 'long unsigned int'} [-Wsign-compare]
     for(int i=0;i<s.length();i++){
                  ~^~~~~~~~~~
```

Run Time

Input (stdin)

```
(a+b)*c*d
```

Your Output (stdout)

```
ab+c*d*
```

Expected Output

```
ab+c*d*
```

Testcase 0 ✔ | Testcase 1 ✔ | Testcase 2 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Compile Time

Compile Message

```
Solution.cpp: In function 'int main()':
Solution.cpp:28:18: warning: comparison of integer expressions of different signedness: 'int' and 'std::__cxx11::basic_string<char
>::size_type' {aka 'long unsigned int'} [-Wsign-compare]
     for(int i=0;i<s.length();i++){
                  ~^~~~~~~~~~
```

Run Time

Input (stdin)

```
a+b*c
```

Your Output (stdout)

```
abc*+
```

Expected Output

```
abc*+
```

Aim: You are given an array of strings tokens that represents an arithmetic expression in Reverse polish notation.

Program:

```
class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        stack<int> st;
        for(int i=0;i<tokens.size();i++){
            if(tokens[i]=="+" || tokens[i]=="-" || tokens[i]=="*" || tokens[i]=="/"){
                int b=st.top();
                st.pop();
                int a=st.top();
                st.pop();
                switch(tokens[i][0]){
                    case '+':
                        st.push(a+b);
                        break;
                    case '-':
                        st.push(a-b);
                        break;
                    case '*':
                        st.push(a*b);
                        break;
                    case '/':
                        st.push(a/b);
                        break;
                }

            }
            else{
                st.push(stoi(tokens[i]));
            }
        }
        return st.top();
    }
};
```

Input & Output:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tokens =
["2","1","+","3","*"]

Output

9

Expected

9

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tokens =
["4","13","5","/","+"]

Output

6

Expected

6

Contribute a testcase

Aim: Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:
Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

Program:

```
class Solution {
    public boolean isValid(String s) {
        char a[]=new char[s.length()];
        int count=0;
        if(s.length()%2==0){
            for(int i=0;i<s.length();i++){
                if(s.charAt(i)=='(' || s.charAt(i)=='{' || s.charAt(i)=='['){
                    // System.out.println(s.charAt(i));
                    if(s.charAt(i)=='('){
```

```
                a[count]=')';
            }
            else if(s.charAt(i)=='{'){
                a[count]='}';
            }
            else{
                a[count]=']';
            }

            count++;
        }
        else{
            // System.out.println(s.charAt(i));
            if(count==0){
                return false;
            }
            if(a[count-1]==s.charAt(i)){
                // System.out.println("if");
                // System.out.println(s.charAt(i));
                count--;
            }
            else{
                // System.out.println("else");
                // System.out.println(s.charAt(i));
                return false;
            }
        }
    }

    if(count==0){
        return true;
    }
}
else{
    return false;
}
return false;
}
}
```

Input & output:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =
"()"

Output

true

Expected

true

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =
"()[]{}"

Output

true

Expected

true

Contribute a testcase

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =
"(]"

Output

false

Expected

false

Contribute a testcase

Conclusion: Infix notation is the notation in which operators come between the required operands. Postfix notation is the type of notation in which operator comes after the operand. Infix expression can be converted to postfix expression using stack.