**Submitted By:- Ravi Pandey (075BCT065)**

Write an algorithm and program for the following:

1. Perform push and pop operations in stack.

A)Push Operation

The algorithm to perform the push operation in stack is given below:

Push(Stack, MaxN, Data,Top)

If Top=N-1

Print “Stack Overflow!” and return null

Set Top=Top+1

Stack[Top]=Data

Return

B)Pop Operation

The algorithm to perform the pop operation in stack is given below:

Pop(Stack, Item, Top)

If TOP= -1 //stack is empty

Print “Underflow!” and return null

Set Item=Stack[Top]

Set Top=Top-1

Return

Program Code

#include <iostream>

using namespace std;

template <class t>

class Stack

{

  static const int max =5;

  int top=-1;

  t stack[max];

  public:

  bool isFull()

  {

    if(top==max)

        return true;

    return false;

  }

  bool isEmpty()

  {

    if(top==-1)

        return true;

    return false;

  }

  t pop()

  {

      t item;

      if(!isEmpty())

      {

          item = stack[top];

          top--;

      }

      else

      cout<<"Stack is empty!\nYou must add an item to proceed.";

      return item;

  }

  void push(t data)

  {

      if(!isFull())

      {

          top++;

          stack[top]=data;

      }

      else

      cout<<"Stack is Full!\nYou must pop an item to proceed.";

  }

  void showStack()

  {

      cout<<"\nStack:";

      for(int i=0;i<=top;i++)

      cout<<endl<<i<<"="<<stack[i];

      cout<<"\n................................\n";

  }

};

int main()

{

    //cout<<"Enter type data to be entered:\n1)Number\n2)Character";

    Stack <int> s;

    int item;

    char choice,op;

    do

    {

    cout<<"Which operation do you want to perform?\n1)Push\n2)Pop";

    cin>>choice;

    switch(choice)

    {

        case '1':

        {

            cout<<"Enter number to be pushed:";

            cin>>item;

            s.push(item);

            break;

        }

        case '2':

        {

            cout<<s.pop()<<" Popped";

            break;

        }

        default:

        {

            cout<<"Invalid Operation!";

            op='y';

        }

    }

    cout<<"Do you want to continue?(Y or N)";

    cin>>op;

    s.showStack();

    } while (toupper(op)=='Y');

    return 0;

}

2. Convert :

a. Infix expression to postfix expression

1. START
2. Scan the infix expression form left to right
3. If the scanned character is an operand, then add it to postfix expression
4. The scanned character is an operator then:
5. If the operator currently at the top of the stack has lower precedence than the operator read or the stack is empty then push the operator to the stack
6. Else pop the stack and add the popped operator to postfix expression and push the read operator to the stack
7. If stack is not empty after reading all characters from infix expression then pop operators from stack to postfix expression until the stack is empty
8. Display the resulting postfix expression
9. END

b. Infix expression to prefix expression

1. START
2. Reverse the given infix expression
3. Interchange all ‘(‘ into ‘)’ and vice-versa.
4. Convert the obtained expression into postfix expression
5. Reverse the obtained expression so that it becomes the required prefix expression.
6. END

Program Code

#include<bits/stdc++.h>

using namespace std;

int precedence(char optr)

{

    if(optr=='^')

        return 3;

    else if(optr=='\*' || optr=='/')

        return 2;

    else if (optr=='+' || optr=='-')

        return 1;

    else

        return -1;

}

string infixToPostfix(string Expr)

{

    stack<char> st;

    string postExpr;

    int length=Expr.length();

    st.push('N');

    for(int i=0;i<length;i++)

    {

        if(Expr[i]>='a' && Expr[i]<='z')

        postExpr+=Expr[i];

        else if(Expr[i]==')')

        {

            do

            {

                postExpr=postExpr+st.top();

                st.pop();

                if(st.top()=='(')

                    st.pop();

            }while(st.top()!='(' && st.top()!='N');

        }

        else if(Expr[i]=='(')

        st.push('(');

        else

        {

            if(precedence(Expr[i])>precedence(st.top()))

            st.push(Expr[i]);

            else

            {

            postExpr=postExpr+st.top();

            st.pop();

            st.push(Expr[i]);

            }

        }

    }

    return postExpr;

}

string infixToPrefix(string Expr)

{

    reverse(Expr.begin(),Expr.end());

    for(int i=0;i<Expr.length();i++)

    {

        if (Expr[i] == '(') {

            Expr[i] = ')';

            i++;

        }

        else if (Expr[i] == ')') {

            Expr[i] = '(';

            i++;

        }

    }

    string revExpr = infixToPostfix(Expr);

    reverse(revExpr.begin(),revExpr.end());

    return revExpr;

}

int main()

{

    string Expr =  "(a-b/c)\*(d/e-f)";

    cout<<"Postfix: "<<infixToPostfix(Expr)<<"\nPrefix: "<<infixToPrefix(Expr);

    return 0;

}

3. Evaluate the postfix expression

1. START
2. Scan each character of the postfix expression from left to right
3. If operand is encountered, push it onto Stack

[End If]

1. If operator is encountered
2. A -> Top element and pop
3. B-> Next to Top element and pop
4. Evaluate B operator A
5. push result of B operator A onto Stack
6. Set result = stack[top]
7. END

Program Code

#include<bits/stdc++.h>

using namespace std;

float getRes(float a, float b, char op)

{

    switch(op)

            {

                case '+':

                    return b+a;

                case '-':

                    return b-a;

                case '\*':

                    return b\*a;

                case '/':

                    return b/a;

                default:

                {

                cout<<"error!";

                exit(0);

                }

            }

}

float evalPostfix(string expr)

{

    stack <char> st;

    float a,b;

    float res;

    for(int i=0;i<expr.length();i++)

    {

        if(expr[i]>='0' && expr[i]<='9')

        {

            st.push(expr[i]);

        }

        else if(expr[i]=='+' ||expr[i]=='-' ||expr[i]=='\*' ||expr[i]=='/')

        {

            a=static\_cast<float>(st.top()-48);

            st.pop();

            b=static\_cast<float>(st.top()-48);

            st.pop();

            res=getRes(a,b,expr[i]);

            st.push(static\_cast<char>(res+48));

        }

    }

    return static\_cast<float>(st.top()-48);

}

int main()

{

    float a=0;

    char aa=static\_cast<char>(49);

    //cout<<aa;

    cout<<evalPostfix("456\*+");

return 0;

}

4. Check the paired parenthesis in mathematical expression.

1. START
2. Scan each character of the expression from left to right
3. If left parenthesis is encountered, push it onto Stack
4. If right parenthesis is encountered
5. And stack is not empty i.e it contains one left parenthesis then pop it out.
6. And stack is empty then print “Unbalanced Parentheses” and exit
7. If stack is empty after scanning all the characters then

Print “Balanced parentheses”

1. END

Program Code

#include <bits/stdc++.h>

using namespace std;

bool checkParan(string expr)

{

    stack <char> st;

    st.push('T');

    for(int i=0;i<expr.length();i++)

    {

        if(expr[i]=='(')

        st.push('(');

        else if(expr[i]==')')

        {

            if(st.top()!='T')

            st.pop();

            else

            return false;

        }

    }

    if(st.top()=='T')

    return true;

    else

    return false;

}

int main()

{

    string str;

    cout<<"Enter expression:";

    cin>>str;

    if(checkParan(str))

    {

        cout<<"All paranthesis are paired";

    }

    else

    {

        cout<<"Unpaired paranthesis.";

    }

    return 0;

}