Name - Srushti Bhivaji Salgar
PRN - B24CE1079
Class - SE 2 Batch - A
Subject - Data Structures
Assignment 5

/*PROBLEM STATEMENT:
Parenthesis Checker:
Write a program using a stack for push, pop, peek, and isEmpty operations. Write isBalanced() Function that Iterates through the input expression, Pushes opening brackets onto the stack. For closing brackets, it checks the top of the stack for a matching opening bracket. Ensures that all opening brackets are matched by the end of the traversal. Main Function: Accepts a string expression from the user. Uses isBalanced() to determine if the parentheses in the expression are balanced.*/

```
#include<iostream>
using namespace std;

class checker{
        int n=20;
        char arr[20];
        int top = -1;

        public:
        int isEmpty();
        int isFull();
        void PUSH(char x);
        char POP();
        char PEEK();

};

int checker::isEmpty(){
        if(top ==-1){
                return 1;
        } else{
                return 0;
        }
}

int checker:: isFull(){
        if(top == n-1){
                return 1;
```

```
        } else{
                return 0;
        }
}


void checker::PUSH(char x){
        if(!isFull()){
                top = top+1;
                arr[top]=x;
        }
}

char checker::POP(){
        char x = '\0';
        if(!isEmpty()){
                x=arr[top];
                top--;
        }
        return x ;
}
 char checker::PEEK(){
         if(!isEmpty()){
                 return arr[top];
         }
         return '\0';
 }
 int main() {
    checker s;
    char exp[20];
    cout << "Enter an expression (max 19 characters): ";
    cin >> exp;

    cout << "Entered expression: ";
    for (int i = 0; exp[i] != '\0'; i++) {
        cout << exp[i] << " ";
    }
    cout << endl;

    for (int i = 0; exp[i] != '\0'; i++) {
        if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{') {
            s.PUSH(exp[i]);
        } else if (exp[i] == ')' || exp[i] == ']' || exp[i] == '}') {
            if (s.isEmpty()) {
```

```
            cout << "Parentheses are not balanced." << endl;
            return 0;
        } else {
            char c = s.PEEK();
            if ((exp[i] == ')' && c == '(') ||
                (exp[i] == ']' && c == '[') ||
                (exp[i] == '}' && c == '{')) {
                s.POP();
            } else {
                cout << "Parentheses are not balanced." << endl;
                return 0;
            }
        }
    }
  }


  if (!s.isEmpty()) {
     cout << "Parentheses are not balanced." << endl;
  }
  else{
  cout << "Parentheses are balanced." << endl;

}
return 0;
}
```

OUTPUT

```
Enter an expression (max 19 characters): 2*5+(3/1+[7-{9*4}])
Entered expression: 2 * 5 + ( 3 / 1 + [ 7 - { 9 * 4 } ] )
Parentheses are balanced.
```

```
Enter an expression (max 19 characters): 6*5{79+4[5-1(2
Entered expression: 6 * 5 { 7 9 + 4 [ 5 - 1 ( 2
Parentheses are not balanced.
```

/***Syntax Parsing in Programming Languages:**
Parsing expressions is a key step in many compilers and language processors. When a language's syntax requires parsing mathematical or logical expressions, converting between infix and postfix notation ensures that expressions are evaluated correctly. Accept an infix expression and show the expression in postfix form.*/

```
#include <iostream>
#include <cstring>
#include <string>
#define N 10
using namespace std;
class stack
{
        public:
 char arr[10];
 string ex;
 int top;
        stack()
        {
        top=-1;
        }
        void push(char c)
        {
        if(top==(N-1))
        {cout<<"stack overflow";}
        else{top+=1;
        arr[top]=c;}
        }
        char pop()
        {   char c= ' ';
        if(top==-1)
        {//cout<<"Stack underflow";
        }
        else
        {
        c=arr[top];
```

```
        top --;
        }
        return c;
        }
        int precedence(char opr){
        if (opr=='*' || opr=='/')
        return 2;
        if (opr=='+' || opr=='-')
        return 1;
        if(opr=='(')  return 0;
        }
        char associativity(char opr){
        if (opr=='*' || opr=='/'||opr=='+' || opr=='-')
        return 'L';
        else
        return 'R';
        }
        char peek(){return arr[top]; }
        string InfixToPostfixConversion(string ex);
};

string stack::InfixToPostfixConversion(string ex){
        int l=ex.length();
        int i=0,j=0;
        char op_exp[20];
        char ch,ch1;
        while(i<l){

        //cout<<ex[i];
        if(ex[i]=='+' ||ex[i]=='-'||ex[i]=='*' || ex[i]=='/'){

        if(top==-1) push(ex[i]);
        else{
                ch=peek();
                //if(ch== ' ')
                while(precedence(ex[i])<=precedence(ch)){
                ch1=pop();
```

```
            op_exp[j++]=ch1;
            ch=peek();
            }
            push(ex[i]);
        }
        }
        else if(ex[i]=='(')
        push(ex[i]);
        else if(ex[i]==')'){
        ch1=pop();
        while(ch1!='('){
            op_exp[j++]=ch1;
            ch1=pop();
        }
        }
        else  {
        op_exp[j++]=ex[i];
        // var++;
        }
        i++;

        }
        do{
        ch=pop();
        op_exp[j++]=ch;
        }while(ch!=' ');
        op_exp[j]='\0';
        return op_exp;

        }

int main() {
 stack s;
 string ex;
 cout<<"\n Enter an expression:";
 cin>>ex;
 string op_exp=s.InfixToPostfixConversion(ex);
```

```
cout<<"\n Postfix Expression is: "<<op_exp;
return 0;

}
```

OUTPUT

```
Enter an expression:a+b*c/d

Postfix Expression is: abc*d/+
```

```
Enter an expression:a+b*(c-d-e)*(f+g*h)-i

Postfix Expression is: abcd-e-*fgh*+*+i-
```