ASSIGNMENT No. 05

TITLE: Parenthesis Checker OR Syntax Parsing in Programming Languages
Name: Munajja Mujafar Dalimbkar
PRN: B25CCE2011

PROBLEM STATEMENT:

Parenthesis Checker:

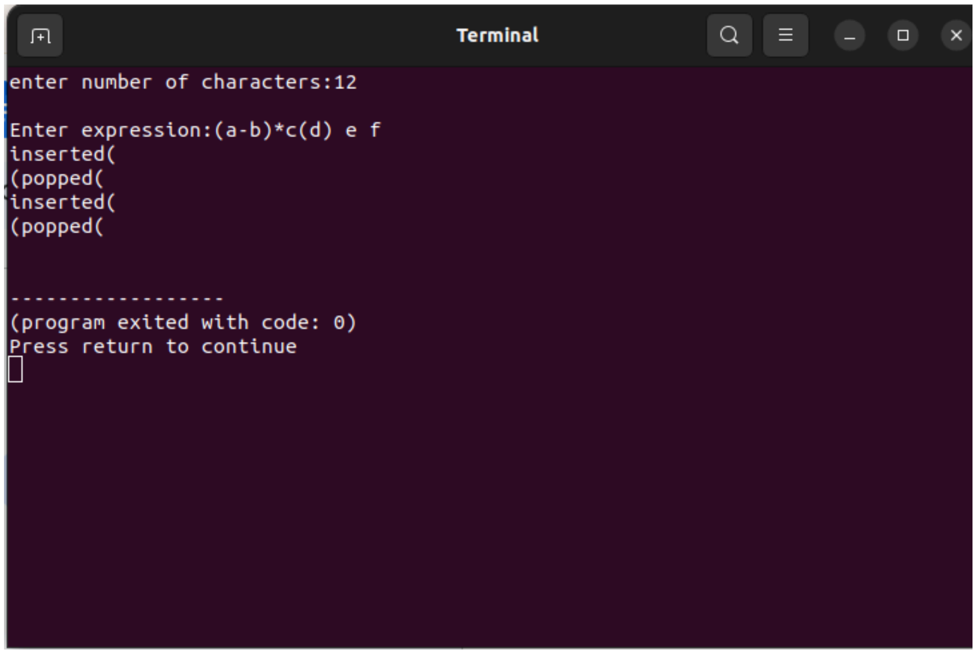
Write a program using a stack for push, pop, peek, and isEmpty operations. Write isBalanced() Function that Iterates through the input expression, Pushes opening brackets onto the stack. For closing brackets, it checks the top of the stack for a matching opening bracket. Ensures that all opening brackets 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.

CODE:

```
#include<iostream>
using namespace std;
class parenthesis_checker{
char stack[100];
int N=5;
char x;
int top=-1;
public:
bool isFull();
bool isEmpty();
void push(char x);
char pop();
void display();
char peek();
};

bool parenthesis_checker::isFull(){
if(top==N-1){
return true;
}
else{
return false;
}
}
bool parenthesis_checker::isEmpty(){
if(top==-1){
return true;
}
else{
return false;
}
}
void parenthesis_checker::push(char x){
if(isFull()){
cout<<"stack is full!"<<endl;
}
else{
top++;
stack[top]=x;
cout<<"inserted"<<stack[top]<<endl;
}
}
char parenthesis_checker::pop(){
if(isEmpty()){
cout<<"stack is empty"<<endl;
}
else{
x=stack[top];
cout<<"popped"<<x<<endl;
top--;
}
return x;
}
char parenthesis_checker::peek(){
cout<<stack[top];
return stack[top];
}
int main(){
parenthesis_checker s;
char x;
int n;
cout<<"enter number of characters:";
```

```
cin>>n;
char exp[n];

cout<<"\nEnter expression:";
cin>>exp;

for(int i=0;i<n;i++){
if(exp[i]=='{'||exp[i]=='['||exp[i]=='('){
s.push(exp[i]);
x=s.peek();
}
else if( (exp[i]=='}' && x=='{') || (exp[i]==')' && x=='(') || (exp[i]==']' && x=='[')){
x=s.pop();
}
}

return 0;
}
```

0utput:



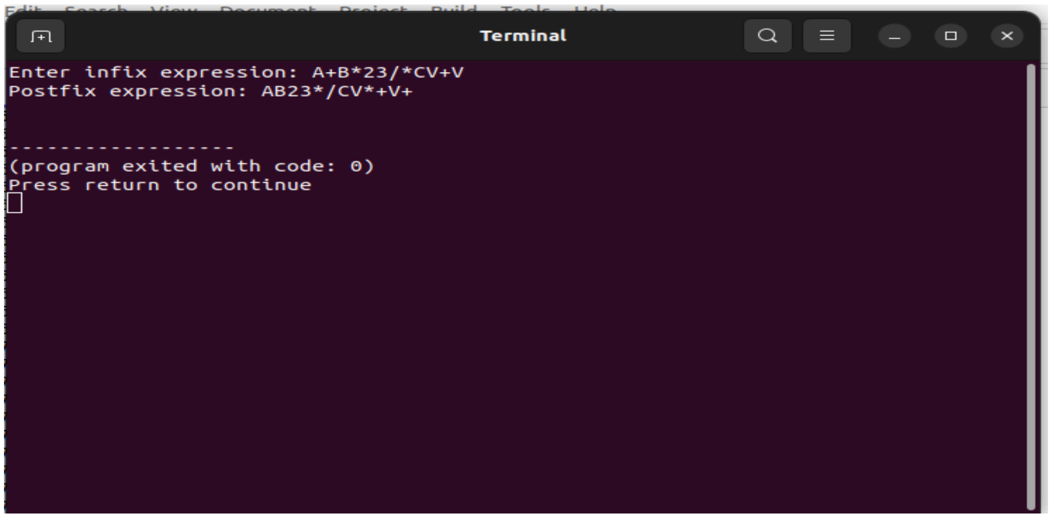
```
enter number of characters:12

Enter expression:(a-b)*c(d) e f
inserted(
(popped(
inserted(
(popped(


------------------
(program exited with code: 0)
Press return to continue
```

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.

CODE:
```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Function to return precedence of operators
int precedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    if (op == '^')
        return 3;
    return 0;
}

// Function to check if the character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

// Function to convert infix to postfix
string infixToPostfix(const string& infix) {
    stack<char> s;
    string postfix = "";

    for (char c : infix) {
        // If operand, add to output
        if (isalnum(c)) {
            postfix += c;
        }
        // If '(', push it
        else if (c == '(') {
            s.push(c);
        }
        // If ')', pop until '(' is found
        else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            if (!s.empty())
                s.pop();  // Remove '('
            else {
                cout << "Error: Mismatched parentheses\n";
                return "";
            }
        }
        // If operator
        else if (isOperator(c)) {
            while (!s.empty() && precedence(s.top()) >= precedence(c) && s.top() != '(') {
                // For right-associative operator '^', use > instead of >=
                if (c == '^' && s.top() == '^')
                    break;
                postfix += s.top();
                s.pop();
            }
            s.push(c);
        }
        // Ignore spaces
        else if (c == ' ' || c == '\t') {
            continue;
        }
        else {
            cout << "Error: Invalid character '" << c << "' in expression\n";
            return "";
        }
    }

    // Pop remaining operators
    while (!s.empty()) {
        if (s.top() == '(') {
            cout << "Error: Mismatched parentheses\n";
            return "";
        }
        postfix += s.top();
        s.pop();
    }

    return postfix;
}
```

```
int main() {
    string infixExp;
    cout << "Enter infix expression: ";
    getline(cin, infixExp);

    string postfixExp = infixToPostfix(infixExp);
    if (!postfixExp.empty()) {
        cout << "Postfix expression: " << postfixExp << "\n";
    }

    return 0;
}
```

OUTPUT:



```
Enter infix expression: A+B*23/*CV+V
Postfix expression: AB23*/CV*+V+


------------------
(program exited with code: 0)
Press return to continue
```