T.Y.B.Tech (CSE)

System Software and Compilers(SSC)

Lab Assignment No – 6

Name: Aniruddha Shende

Roll number: PD-05

Batch: D1

Panel: D

Name:- Aniruddha Arun Shende
Roll no:- PD05
Batch:- D1
Panel: D

MIT-WPU MIT WORLD PEACE UNIVERSITY | PUNE

Subject :- SSC

Lab Assignment No-6 RDP

Title:- Implement recursive descent parser for sample language.

Aim :- Implement recursive descent parser for sample grammar.

Objective:-
1. To study parsing phase in the compiler
2. To study types of parsers - top down & bottom-up.
3. Problems encountered during top down parser.
4. How to write a top down parser.

Theory:-

1. CFG, non-terminals, productions, derivation sequence.

Ans-1. A context-free grammar G=(T, N, S, P) consists of:
a. T, set of Terminals
b. N, set of Non-terminals
c. S, a designated start nonterminal.
d. P, set of production.

N, a set of nonterminals (syntactic variables generated by productions).
P is a set of productions & each production has the form, A ::= α, where A ⇒ nonterminal, α ⇒ sentential form. A parse tree is a graphical sequence of a sentential form.

2. Introduction to Recursive Descent Parser

Ans·2: A parser that uses a set of recursive procedures to recognize its input is called a recursive descent parsing.
It is an attempt to find the leftmost derivation for an input string.

3. Elimination of Left recursion.

Ans·3: A grammar is left recursive if it has a NT A such that there is a derivation $A \Rightarrow^{+} A\alpha$ for some string α.
Top down parsing methods cannot handle left recursive grammars, so a transformation that eliminates left recursion is needed.
Eg:- $A \rightarrow A\alpha \mid \beta$
It has left recursion. To eliminate it we rewrite it as
$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \epsilon$

MIT-WPU MIT WORLD PEACE UNIVERSITY | PUNE

4. Give Example:-

Ans 4: $A \rightarrow A\alpha | \beta$. It has left recursion. To eliminate it we rewrite it as

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' | \epsilon$

The technique to eliminate left recursion is:

$A \rightarrow A\alpha 1 | A\alpha 2 | \ldots | \beta 1 | \beta 2 | \ldots | \beta n$

no βi begins with A.

So we replace the A-productions by

$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$

$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \epsilon$

This is left-recursive grammar:

$E \rightarrow E+T | T$

$T \rightarrow T*F | F$

$F \rightarrow (E) | id$

Eliminate the immediate left recursion.

★ Input:- String satisfying given grammar, string not satisfying given grammar to test error condition.

★ Output:- Success for correct string, failure for syntactically wrong string.

* Conclusion :- The recursive descent parser is successfully implemented.

* Platforms :- Linux (C/C++/JAVA)

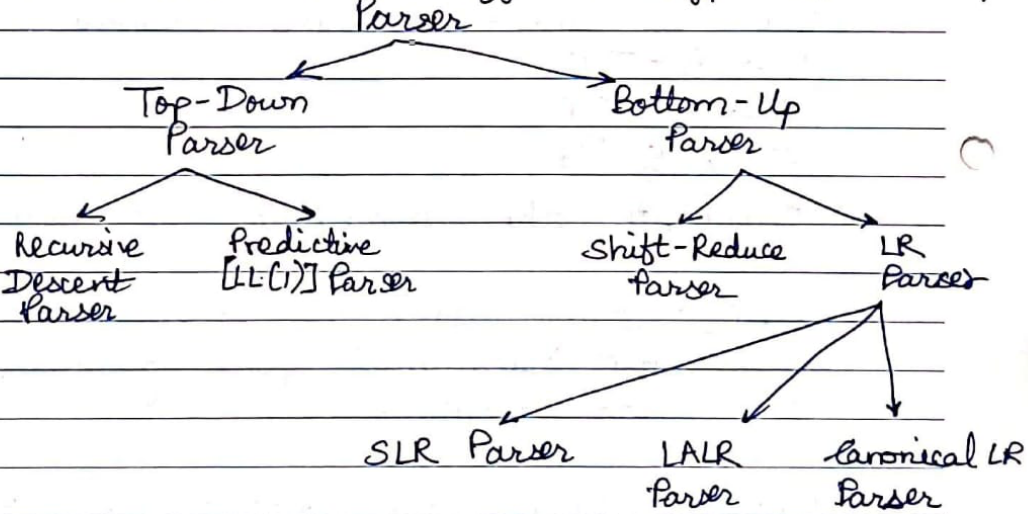
FAQ's :-

1) What is Parsing?

Ans 1) Parsing is the process of analyzing a string of symbols, either in Natural Language, computer languages or data structures, conforming to the rules of a formal grammar. Used to determine if start symbol can derive the program or not.

2) What are the different types of Parser?

Ans 2)



3) What are the disadvantages of RDP?

Ans 3) Following are some of the disadvantages of RDP:-

(a) They are not as fast as some other methods.

(b) It is difficult to provide good error messages.

(c) They cannot do parses that require arbitrarily long lookaheads.

4) Why to eliminate the left recursion?

Ans 4) If we do not remove left recursion, then a top-down parser might loop forever when parsing an expression using this grammar.

Java Code & Output:

```
#include <bits/stdc++.h>

using namespace std;

string s;
char tk;
int pt = 0;

void E();
void ED();
void T();
void TD();
void F();

void advance()
{
    pt++;
    tk = s[pt];
}

void E()
{
    T();
    ED();
}

void ED()
{
    if (tk == '+')
    {
        advance();
        T();
        E();
    }
}

void T()
{
```

```
    F();
    TD();
}

void TD()
{
    if (tk == '*')
    {
        advance();
        F();
        T();
    }
}

void F()
{
    if (tk == '(')
    {
        advance();
        E();
        if (tk == ')')
        {
            advance();
        }
    }
    else
    {
        if (tk == 'i')
        {
            advance();
        }
    }
}

int main()
{

    while (1)
    {
        cout << "\nEnter the string: ";
        cin >> s;
        tk = s[pt];
```

```
        E();
        if (s[pt] == '\0')
            cout << "String is ACCEPTED\n";
        else
            cout << "String is NOT ACCEPTED\n";
        int choice;
        cout << "Do you want to continue? (1--yes |
0--no )" << endl;
        cin >> choice;
        if (choice == 0)
            exit(0);
        pt = 0;
    }
    return 0;
}
```

Output:

```
ani@Aniruddhas-MacBook-Pro Lab Assign 6 % cd "/Users/ani/Desktop/Tri-9/SSC/Lab/Lab A
ssign 6/" && g++ --std=c++17 rdp1.cpp -o rdp1 && "/Users/ani/Desktop/Tri-9/SSC/Lab/L
ab Assign 6/"rdp1

Enter the string: i*i
String is ACCEPTED
Do you want to continue? (1--yes |  0--no )
1

Enter the string: (i+i)*i
String is ACCEPTED
Do you want to continue? (1--yes |  0--no )
0
```