SSC LAB ASSIGNMENT NO.6

RECURSIVE DECENT PARSER

NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

Title: Implementing recursive descent parser for sample language.

Aim: Implement Recursive Descent parser for grammar of Arithmetic expression.

Objective:

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

Theory:

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

CFG − A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N, T, P, S) where

- N is a set of non-terminal symbols.
- T is a set of terminals where N ∩ T = NULL.
- P is a set of rules, P: N → (N ∪ T)*, i.e., the left-hand side of the production rule P does have any right context or left context.
- S is the start symbol.

A context-free grammar has four components:

- A set of non-terminals (V). Non-terminals are syntactic variables that denote sets of strings. The non-terminals define sets of strings that help define the language generated by the grammar.
- A set of tokens, known as terminal symbols (Σ). Terminals are the basic symbols from which strings are formed.
- A set of productions (P). The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a non-terminal called the left side of the production, an arrow, and a sequence of tokens and/or on- terminals, called the right side of the production.

- One of the non-terminals is designated as the start symbol (S); from where the production begins.

The strings are derived from the start symbol by repeatedly replacing a non-terminal (initially the start symbol) by the right side of a production, for that non-terminal.

Example

We take the problem of palindrome language, which cannot be described by means of Regular Expression. That is, L = { w | w = wR } is not a regular language. But it can be described by means of CFG, as illustrated below:

G = (V, Σ, P, S)

Where:

V = { Q, Z, N }

Σ = { 0, 1 }

P = { Q → Z | Q → N | Q → ℇ | Z → 0Q0 | N → 1Q1 }

S = { Q }

This grammar describes palindrome language, such as: 1001, 11100111, 00100, 1010101, 11111, etc.

2. Introduction to Recursive Descent Parser.

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as predictive parsing.

This parsing technique is regarded recursive as it uses context-free grammar which is recursive in nature.

3. Elimination of Left recursion.

A production of grammar is said to have left recursion if the leftmost variable of its RHS is same as variable of its LHS.A grammar containing a production having left recursion is called as Left Recursive Grammar.

Left recursion is eliminated by converting the grammar into a right recursive grammar.

If we have the left-recursive pair of productions-
A → Aα / β
(Left Recursive Grammar)
where β does not begin with an A.

Then, we can eliminate left recursion by replacing the pair of productions with-
A → βA'
A' → αA' / ∈
(Right Recursive Grammar)

This right recursive grammar functions are the same as left recursive grammar.

4. Give Example:

This left-recursive grammar:

E → E + T | T

T → T * F | F

F → (E) |i

Eliminate the immediate left recursion.

E → T E'

E' → +T E' | Epsilon

T → FT'

T' → *FT' | Epsilon

F → (E) | id

Recursive procedures to recognize arithmetic expressions

```
Procedure E( ):
     begin
               T( )
               E’  ( )
     End;
procedure E’( ):
          If input_symbol = ‘+’ then
          begin
               ADVANCE( )
               T( )
               E’( )
          end;
procedure T( ) :
         begin
               F( )
               T’( )
     End;
procedure T’( )
          If input_symbol = ‘*’ then
          begin
          ADVANCE( )
               F( )
               T’( )
          End;

 procedure F( ):
          If input_symbol = ‘i’ then
          ADVANCE( )
     Else if input Symbol=‘ ( ’ then
          begin
          ADVANCE( )
          E( )
     If input_symbol = ‘ ) ’ then
          ADVANCE( )
          else ERROR( )
     end
else ERROR( )
```

NOTE: ADVANCE( ) moves the input pointer to the next input symbol

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

Output: Success for correct string, Failure for syntactically wrong string.

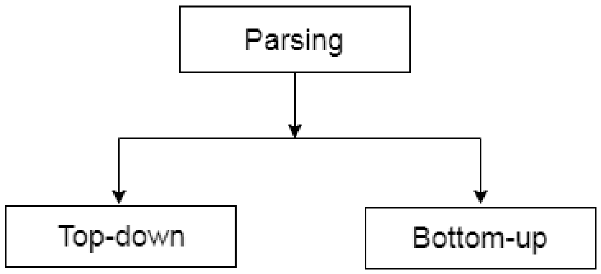
Conclusion: The recursive descent parser is successfully implemented.

Platform: Linux (C/C++/JAVA)

FAQ's:

1. What is Parsing?

Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase. A parser takes input in the form of sequence of tokens and produces output in the form of parse tree. Parsing is of two types: top down parsing and bottom up parsing.



Top down paring

- The top down parsing is known as recursive parsing or predictive parsing.
- Bottom up parsing is used to construct a parse tree for an input string.
- In the top down parsing, the parsing starts from the start symbol and transform it into the input symbol.

Bottom up parsing

- Bottom up parsing is also known as shift-reduce parsing.
- Bottom up parsing is used to construct a parse tree for an input string.
- In the bottom up parsing, the parsing starts with the input symbol and construct the parse tree up to the start symbol by tracing out the rightmost derivations of string in reverse.

2. What are the different types of Parser?

a. Recursive Descent Parsing
b. Back-tracking
c. Predictive Parser

d. LL Parser
e. LL Parsing Algorithm
f. Shift-Reduce Parsing
g. LR Parser
h. LR Parsing Algorithm

3. What are the disadvantages of RDP?

a. They are not as fast as some other methods.
b. It is difficult to provide really good error messages.
c. They cannot do parses that require arbitrarily long lookaheads.

4. Why eliminate the left recursion?

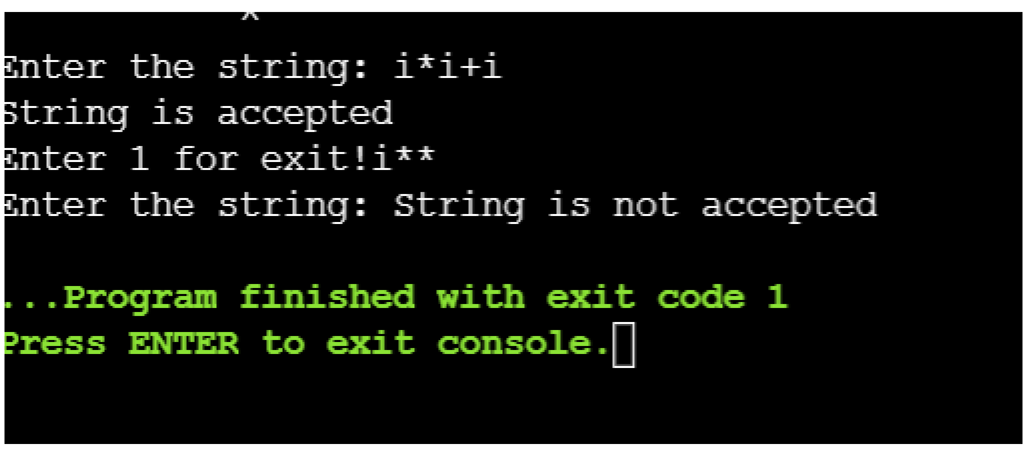
Left recursion often poses problems for parsers, either because it leads them into infinite recursion (as in the case of most top-down parsers) or because they expect rules in a normal form that forbids it (as in the case of many bottom-up parsers, including the CYK algorithm).

PROGRAM :

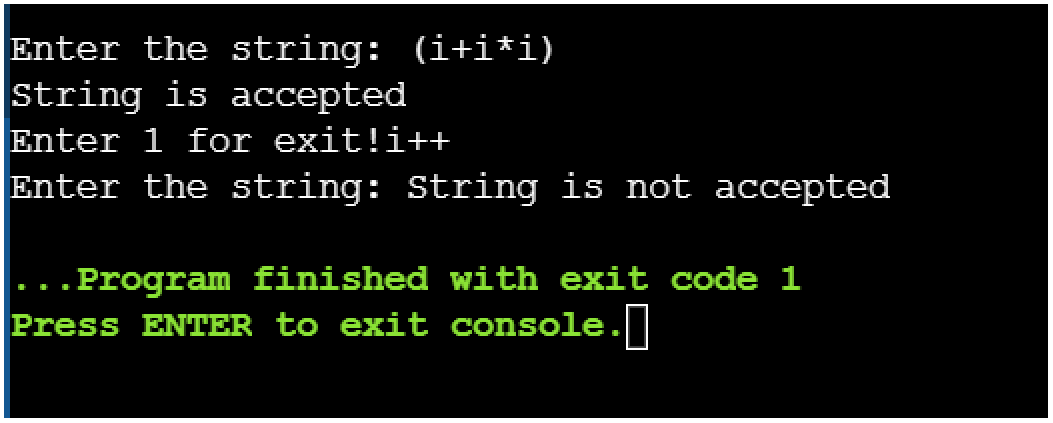
```
/*
 SSC LAB ASSIGNMENT NO.6
RECURSIVE DECENT PARSER
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1
 */

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

void E();
int i=0;
char str[10],tp;
void advance()
{
   i++;
   tp=str[i];
}
void F()
{
   if(tp=='i')
   {
         advance();
```

```
    }
    else
    {
        if(tp=='(')
        {
            advance();
            E();
            if(tp==')')
            {
                advance();
            }
        }
        else
        {
            printf("String is not accepted");
            exit(1);
        }
    }
}
void TP()
{
    if(tp=='*')
    {
        advance();
        F();
        TP();
    }
}

void T()
{
    F();
    TP();
}

void EP()
{
    if(tp=='+')
    {
        advance();
        T();
        EP();
    }
}

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

int main()
{
    int op;
    while(1)
    {
        printf("Enter the string: ");
        scanf("%s",&str);
        tp=str[i];
        E();
        if(tp=='\0')
        {
             printf("String is accepted\n");
        }
        else
        {
             printf("String is not accepted");
        }
        printf("Enter 1 for exit!");
        scanf("%d",&op);
        if(op==1)
        {
             exit(0);
        }
    }
}
```

OUTPUT SS :



```
Enter the string: i*i+i
String is accepted
Enter 1 for exit!i**
Enter the string: String is not accepted

...Program finished with exit code 1
Press ENTER to exit console.
```



```
Enter the string: (i+i*i)
String is accepted
Enter 1 for exit!i++
Enter the string: String is not accepted

...Program finished with exit code 1
Press ENTER to exit console.
```