RECURSIVE DESCENT PARSER FOR A MINIATURE GRAMMAR

Objective:

- To understand implementation of a recursive descent parser for a miniature C grammar.
- To write Context-Free Grammar for parsing

Prerequisites:

- Knowledge of top down parsing.
- Knowledge on removal of left recursion from the grammar.
- Knowledge on performing left factoring on the grammar.
- Understanding about computation of first and follow for a grammar.

I. TOP DOWN PARSERS:

Syntax analysis is the second phase of the compiler. Output of lexical analyzer – tokens and symbol table are taken as input to the syntax analyzer. Syntax analyzer is also called as parser.

Top Down parsers come in two forms

- ✓ Backtracking parsers
- ✓ Predictive parsers

Predictive parsers are categorized into

- ✓ Recursive Descent parsers
- ✓ Table driven or LL(1) parsers

RECURSIVE DESCENT PARSERS:

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.

II. SOLVED EXERCISE:

E→ num T

T→ *num T|ε

Grammar 6.1

[**Note**: Token generated for num is NUMBER]

Sample Code :

```
#include<stdio.h>

#include<ctype.h>

#include “lexel.c”  // lexel.c is lexical analyzer program which is coded in previous lab

#include<string.h>

struct token lookahead;

int i=0;

void proc_t();

void proc_e()

{

lookahead=getNexttoken();

if(strcmp(lookahead.lexemename,”NUMBER”)==0)

{

lookahead=getNexttoken();

proc_t();

}

else

{
```

```
printf("\n Error");
}
}
void proc_t()
  {
lookahead=getNexttoken();
if(strcmp(lookahead.lexemename,”MUL”)==0)
{
lookahead=getNexttoken();
if(strcmp(lookahead.lexemename,”NUMBER”)==0)
{
lookahead=getNexttoken();
            proc_t();
  }
 else
  {
            printf("Error");
   }  } }
  int main()
{
  lookahead=getNextToken();
  proc_e();
if(strcmp(lookahead.lexemename,”EOL”)==0)
```

```
printf("\nSuccessful");

else

printf("\n Error");


 return 0;

 }
```

If the input file contains “NUMBER*NUMBER$”, then token generated is returned by getNextToken( ) which is parsed by the above program. And as input matches the grammar, it displays a success message.

III. LAB EXERCISES:

Write a recursive descent parser for the following simple grammars.

1. S→a | > | (T)
   T→T, S|S

2. E→E+T | T
   T→T*F |F
   F→(E) | id

3. S→aAcBe
   A→Ab|b
   B→d

4. lexp →aterm | list
   aterm →number|identifier
   list→(lexp_seq)
   lexp_seq→ lexp_seqlexp | lexp