Assignment No.6

AIM:

Implement the top-down parsing using recursive decent parsing technique.

THEORY:

Recursive descent parsing

A recursive-descent parsing program consists of a set of procedures, one for each non-terminal. Execution begins with the procedure for the start symbol, which halts and announces success if its procedure body scans the entire input string. Pseudo code for a typical non-terminal appears in Fig. 1. Note that this pseudo code is nondeterministic, since it begins by choosing the A-production to apply in a manner that is not specified.

```
void A() {
        1) Choose an A-production, A ->X1X2 • • Xk
        2) for ( i = 1 to k ) {
        3)      if ( X is a nonterminal )
        4)              Call procedure Xi;
        5)      else if ( Xi equals the current input symbol a )
        6)              Advance the input to the next symbol;
        7)      else /* an error has occurred */;
}
```

Figure 1: A typical procedure for a non-terminal in a top-down parser

General recursive-descent may require backtracking; that is, it may require repeated scans over the input. However, backtracking is rarely needed to parse programming language constructs, so backtracking parsers are not seen frequently.

To allow backtracking, the code of Fig. 1 needs to be modified. First, we cannot choose a unique A-production at line (1), so we must try each of several productions in some order. Then, failure at line (7) is not ultimate failure, but suggests only that we need to return to line (1) and try another A-production. Only if there are no more A-productions to try do we declare that an input error has been found. In order to try another A-production, we need to be able to reset the input pointer to where it was when we first reached line (1). Thus, a local variable is needed to store this input pointer for future use.

Consider the grammar

S->cAd

A->ab|a

To construct a parse tree top-down for the input string w = cad, begin with a tree consisting of a single node labeled S, and the input pointer pointing to c, the first symbol of w. S has only one production, so we use it to expand S and obtain the tree. The leftmost leaf, labeled c, matches the first symbol of input w, so we advance the input pointer to a, the second symbol of w, and consider the next leaf, labeled A. Now, we expand A using the first alternative A -> ab to obtain the tree. We have a match for the second input symbol, a, so we advance the input pointer to d, the third input symbol, and compare d against the next leaf, labeled b. Since b does not match d, we report failure and go back to A to see whether there is another alternative for A that has not been tried, but that might produce a match.

In going back to A, we must reset the input pointer to position 2, the position it had when we first came to A, which means that the procedure for A must store the input pointer in a local variable. The leaf a matches the second symbol of w and the leaf d matches the third symbol. Since we have produced a parse tree for w, we halt and announce successful completion of parsing.

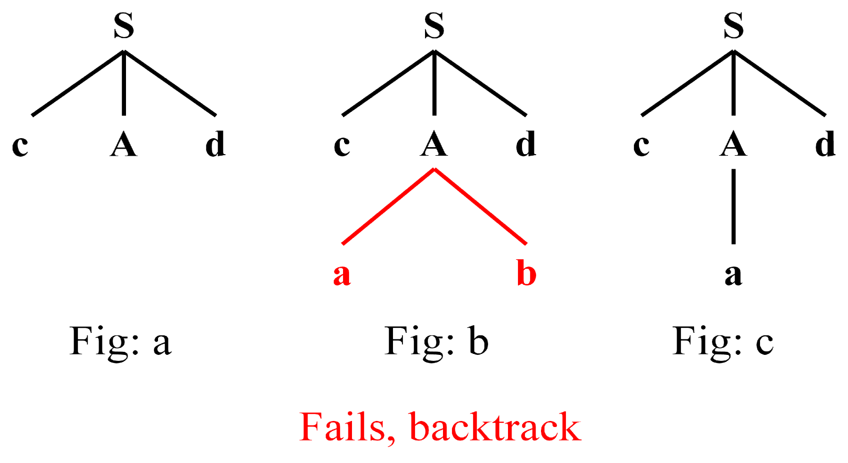

Figure 2: Steps in Top-down parse

PROGRAM:

INPUT:

1. Enter string: cad
2. Enter string: cabd

OUTPUT:

1. cabd
   Backtracking is required
   cad
   Accepted

2. cabd
   Accepted

CONCLUSION:

Thus the recursive descent parsing which requires backtracking is implemented.

REFERENCES:

- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)