Shreya Thigale
TE-IX
33168

PAGE No.
DATE / /

ASSIGNMENT - 9.

TITLE: Recursive descent parser

PROBLEM STATEMENT: Study of Recursive Descent Parser

OBJECTIVE:
1. To understant basic prinples of top-down parsing
2. To study recursive descent parsers.

THEORY:

A recursive descent parser is top down parser, so called because it builds a parse tree from top down, and from left to right, using an input sentence as target as it is scanned from left to right. This type of parser ~~is~~ was very popular for real compilers in past, but is not as popular now. The parser is usually written entirely by hand and does not require any sophisticated tools. It is simple and effective technique, but it is as powerful as some of the shift reduce parsers

The parser uses recursive function corresponding to each grammar rule. For simplicity one can just use the non-terminal as the name of function. The body of recursive function mirrors the right side of corresponding rule. In order of work, one must be able to decide which function to call based on next input symbol.

Consider the grammar:

$E \rightarrow T$
$E \rightarrow T+E$
$T \rightarrow a$
$T \rightarrow a*T$
$T \rightarrow (E)$

We can represent the same grammar shorter:

$E \rightarrow T \mid T+E$
$T \rightarrow a \mid a*T \mid (E)$

An example source string which this language accepts is:
(a)
And is derived as:
$E \rightarrow T \rightarrow (E) \rightarrow (T) \rightarrow (a)$

The recursive descent parsers are widely used on practice, since its very easy to implement them, and the implementation just directly maps the grammar to code.

Shreya Thigale
TE-IX
33168

PAGE No.
DATE / /

ALGORITHM:

1. Start from the main top symbol of grammar. For each non-terminal symbol in grammar implement corresponding parsing function
2. If there are several alternative productions for non-terminal, implement each sub-production as a sub-function and try them in order If some of the sub-productions succeeds, the main non-terminal is considered succeded as well. If sub production does not succeed, then restore the cursor to the beginning of that sub production and try parsing next sub production.
3. Implement function that checks for presence of needed token at the current cursor position in source code. After check it also advances cursor to next token

SAMPLE PROGRAM FOR ABOVE GRAMMAR:

```
~~int main~~()

int   cursor = 0;
int   savecursor = cursor;

function E(){
   return (saveCursor(), E1()) || (backtrack(),
                      saveCursor(), E2());
}
```

Shreya Thigale
TE-IX
33168

PAGE No.
DATE / /

```
function E1() {
    return T();
}

function E2() {
  return T() && term('+') && E();
}

function T() {
  return (saveCursor(), T1()) ||
         (backtrack() , saveCursor(), T2()) ||
         (backtrack() , saveCursor(), T3());
}

function T1() {
   return term('a');
}

function T2() {
   return term('a') && term('*') && T();
}

function T3() {
   return term('(') && E() && term(')');
}

function saveCursor() {
  savedCursor = cursor;
}
```

Shreya Thigale
TE-IX
33168

PAGE No.
DATE / /

```
function backtrack() {
    cursor = saved Cursor;
}

function term (expected) {
    return getNextToken()== expected;
}

function getNextToken () {
    while (source[cursor] == ' ') cursor++;
    char nextToken = source [cursor];
    cursor++;
    return nextToken;
}

char * source;

function parse(s) {
    source = s;
    cursor = 0;
    return E()&& cursor== source.length;
}
```

SAMPLE INPUT	SAMPLE OUTPUT
parse ('(a)');	TRUE
parse ('a*a');	FALSE

Shreya Thigale
TE-IX
33168.

PAGE No.
DATE / /

CONCLUSION:

Thus we have successfully implemented this experimenteo and understood the working of recursive descent parser.