Name - Vasu Kalariya
Roll - PE29
Sub - SSC

Lab Assignment - 6

Title: Implementing recursive descent parser for sample language

Aim: Implemented Recursive descent parser for grammer of Arithmetic expression

Objective:
→ To study parsing phase in the compiler.
→ To study types of parsers - top down and bottom up
→ Problems encountered during top down parser
→ How to write a top down parser

Theory :-

→ CFG, non-terminal, terminals, productions, derivation sequence.

- CFG - A context-free grammer (CFG) consisting of a finite set of grammer rules is a quadruple (N, T, P, S) where,
  N, is a set of non-terminal symbols
  T is a set of terminal symbols
  P is a set of rules, $A \Rightarrow P : N \rightarrow (N \cup T)^*$, i.e the left hand side of the production rule P does have any right context or left context or left context
- S is the start symbol A context-free grammer has four components
- A set of ~~the~~ non-terminals (V). Non-terminals are syntactic variables that denotes set of strings the non-terminals defines sets of strings that help define the language generated by the grammer.
- A set of tokens, knows as terminal symbols (Σ) terminals are the basic symbols from which strings are formed

- A set of ~~tokens~~ productions (P). The productions of a grammer specify the manner in which the terminal and non-terminals can be combined to form strings. Each product^{n} consists of a non-terminal called the left side of the production as arrow, and a sequence of tokens and for non-terminals called the right side of the productions.
- One of the non-terminals is designated as the start ~~s~~ symbol (S); from which the production begins.

→ Introduction to Recursive Descent Parser.

Recursive descent is a top-down parsing technique that consist of the parse tree from the top and the input is read from left to right. It uses procedures for every terminals & non-terminal entity. The parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammer associated with it can't avoid backtracking. A form of recursive-descent parsing that doesn't parsing that doesn't require any backtracking is known as predictive parsing. This parsing technique is requeraded recursive as it uses context free grammer which is recurssive in nature

→ Elimination of left ~~recursive~~ recursion

A production of grammer is said to have left recursion if the leftmost variable of its RHS is a same as variable of its LHS grammer containing a production having left ~~recursio~~e recurrsion is called as left recursi~~ve~~ grammer. Left recursion is eliminations by converting the grammer into a right recurssive grammer.

If we have the left recursive pair of production
$A \rightarrow A\alpha / B$
(left recursive grammer)
where B don't begin with A.
Then, we can eliminate left recursion by replacing the pair of production with
$A \rightarrow BA'$
$A' \rightarrow \alpha A' / E$

Recursive procedure to recognize 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()
```

Input: String ~~st~~ satisfying given grammer, string not satisfying given grammer to test error condition

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

Conclusion: The recursive descent parser is successful implemented.

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

FAQ's

1 What is Parsing?
Parsing is a compiler that is used to break the data into smaller elements ~~coming~~ coming from lexical analysis phase. A parser takes input in the form of sequence of ~~tolerance~~ tokens and produces output in the form of parse tree. Parsing is of two type

(i) Top-down:

→ It is known as recursive parsing or predictive parsing.

→ The parsing starts from the start symbol & transport it into the input symbol.

(ii) Bottom-up

→ Also known as shift-reduce parsing

→ It is used to construct a parse tree for an input string.

2 What are the different types of Parser?

(i) Recursive decent Parsing
(ii) Back tracking
(iii) Predective Parser
(iv) LL Parser
(v) LL Parser Algorithm
(vi) Shift - Reduce Parsing
(vii) LR - Parser
(viii) LR - Parsing Algorithm

3 What are the disadvantages of ROP?

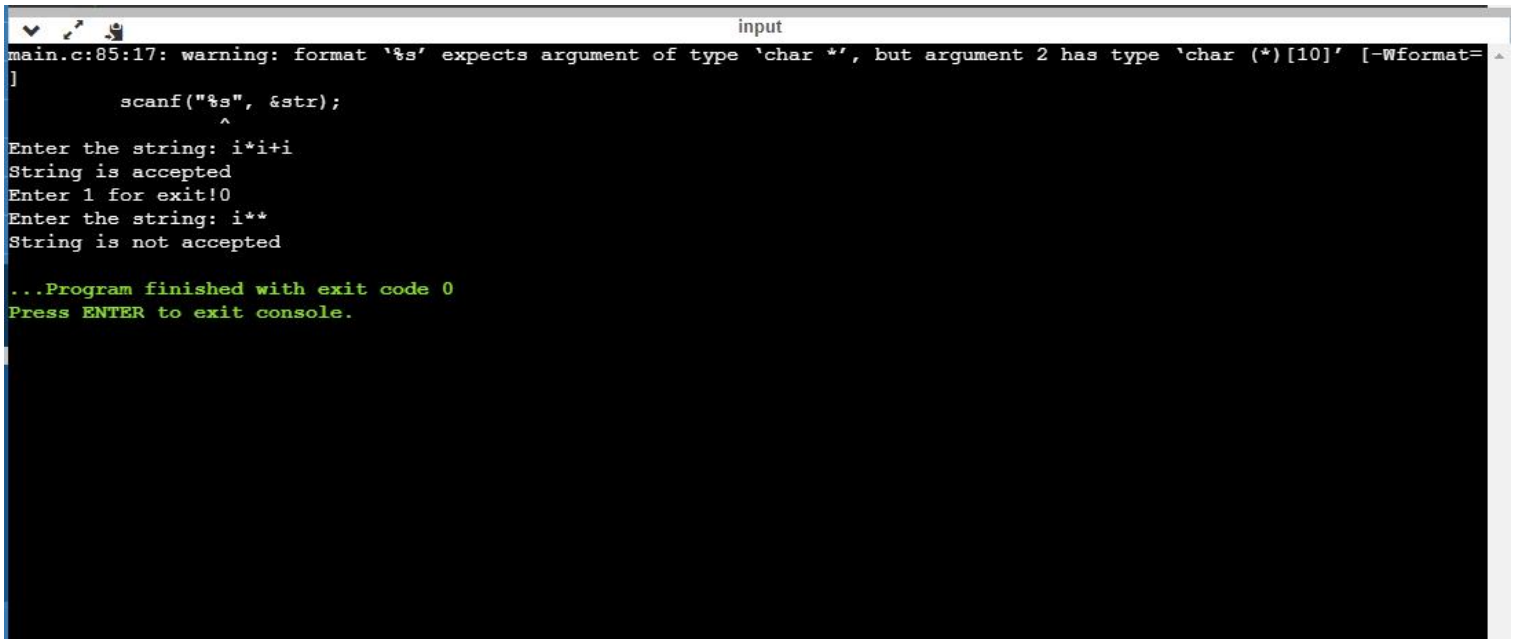
(i) They are not as fast as some other methods
(ii) It is difficult to provide really good error messages.
(iii) They cannot do parses that require arbitory long lookaheads

Q 4 Why eliminate the left recursive?

Left recursive often poses problems for parsers, either because it leads them into infinite ~~recursive~~ recursion or because they expect rules in a normal form that p forbids it.

```c
#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 == 'i')
        {
            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\n");
        }
        printf("Enter 1 for exit!");
        scanf("%d", &op);

        if(op==1)
        {
            exit(0);
        }
    }
}

```


input
main.c:85:17: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[10]' [-Wformat=
]
scanf("%s", &str);
^
Enter the string: i*i+i
String is accepted
Enter 1 for exit!0
Enter the string: i**
String is not accepted
...Program finished with exit code 0
Press ENTER to exit console.