EXPERIMENT - X
RECURSIVE DESCENT PARSER

October 15, 2020

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To construct a recursive descent parser for an expression.

THEORY

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. Recursive descent with backtracking is a technique that determines which production to use by trying each production in turn. Recursive descent with backtracking is not limited to LL(k) grammars, but is not guaranteed to terminate unless the grammar is LL(k). Even when they terminate, parsers that use recursive descent with backtracking may require exponential time.

ALGORITHM

Algorithm 1 Algorithm to construct a Recursive Descent Parser

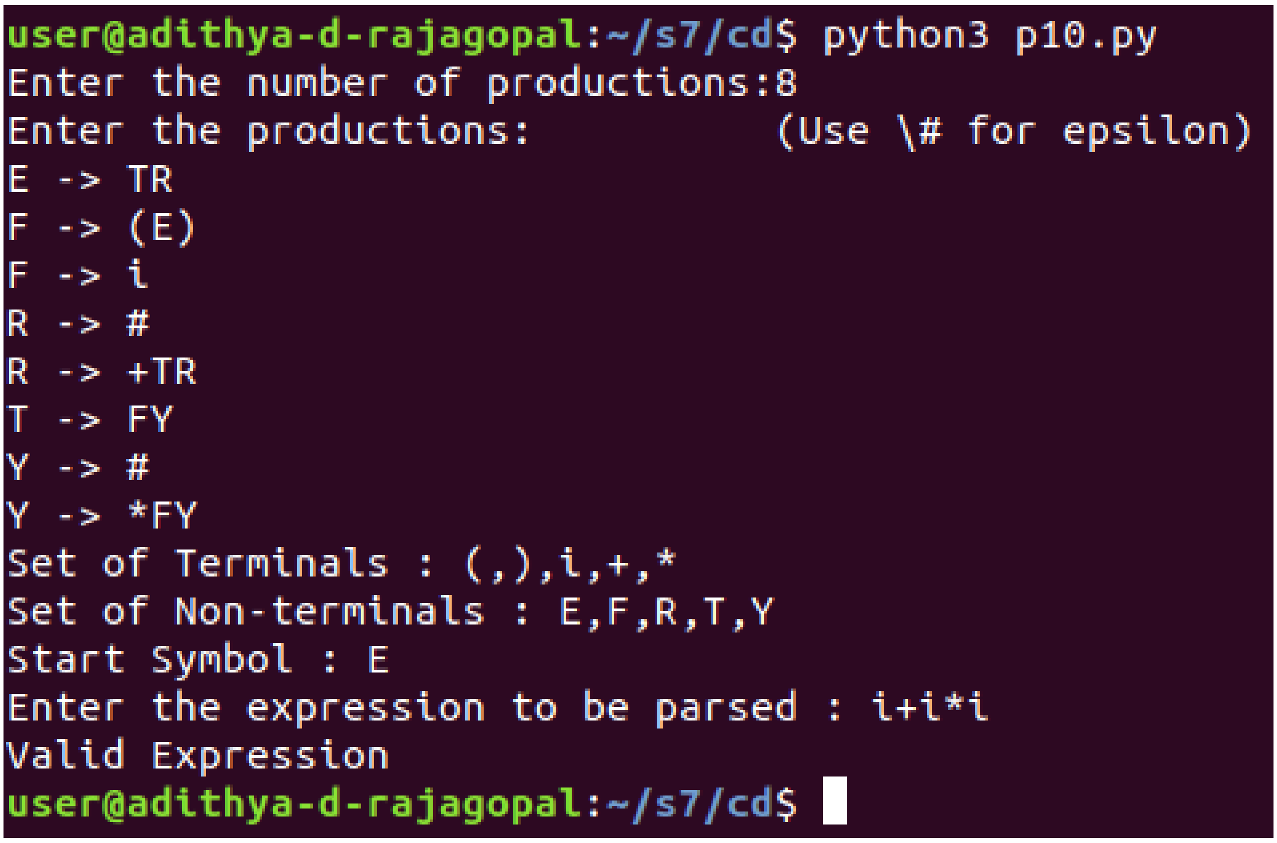
```
1:  Start
2:  Create one parse-method per non-terminal symbol.
3:  procedure A()
4:      Choose a production A → $X_1$ $X_2$ ... $X_k$
5:      for i = 1 to k do
6:          if $X_i$ is a non-terminal then
7:              Call procedure $X_i$.
8:          else if $X_i$ equals the current input symbol a then
9:              advance the input to the next symbol.
10:         else
11:             error();
12:         end if
13:     end for
14: end procedure
15: Stop
```

SOURCE CODE

```
def RDP(pr,str,ip):
        global production
        global T
        global N
        if len(pr)>0 and pr[0]=="#":
                return RDP(pr[1:],str,ip)
        if len(pr)==0:
                if str[ip]=='$':
                        return True
                return False
        if pr[0] in T:
                if str[ip]==pr[0]:
                        return RDP(pr[1:],str,ip+1)
                return False
        if pr[0] in N:
                c=pr[0]
                for i in production[c]:
                        if(RDP(i+pr[1:],str,ip)):
                                return True
        return False

global production
global T
global N
production={}
n=int(input("Enter the number of productions:"))
print("Enter the productions:\t\t(Use \# for epsilon)")
T=[]
N=[]
s=[]
for _ in range(n):
        pr=input().split(" -> ")
        if pr[0] in production.keys():
            production[pr[0]].append(pr[1])
        else:
            production[pr[0]]=[pr[1]]
        if pr[0] not in N:
            N.append(pr[0])
        if pr[0] not in s:
            s.append(pr[0])
        if pr[1]=='#':
            continue
        for i in pr[1]:
            if i not in s:
                s.append(i)
for i in s:
        if i not in N:
            T.append(i)
print("Set of Terminals : ",end="")
for i in range(len(T)-1):
        print(T[i],end=",")
print(T[-1])
print("Set of Non-terminals : ",end="")
for i in range(len(N)-1):
        print(N[i],end=",")
print(N[-1])
S=N[0]
print("Start Symbol :",S)
str=input("Enter the expression to be parsed : ")
w=str+'$'
if(RDP(S,w,0)):
        print("Valid Expression")
else:
        print("Invalid Expression")
```

SAMPLE OUTPUT



```
user@adithya-d-rajagopal:~/s7/cd$ python3 p10.py
Enter the number of productions:8
Enter the productions:          (Use \# for epsilon)
E -> TR
F -> (E)
F -> i
R -> #
R -> +TR
T -> FY
Y -> #
Y -> *FY
Set of Terminals : (,),i,+,*
Set of Non-terminals : E,F,R,T,Y
Start Symbol : E
Enter the expression to be parsed : i+i*i
Valid Expression
user@adithya-d-rajagopal:~/s7/cd$
```

RESULT

A program to construct a recursive descent parser has been implemented using Python and the outputs have been verified.