Name: Niranjan Vinod Patil.
Batch: B31.
Roll No: SCETTYB305.
Course: Compiler Design.

Assignment No: 03

Title:

Recursive Descent Parser.

Aim:

To write c program to implement recursive descent parser which checks whether input string is accepted by given grammar or not.

Theory:

Parsing :

The process of determining if a string of terminals (tokens) can be generated by a grammar. And Parsing is the problem of taking a string of terminal symbols and finding a derivation for that string of symbols in a context-free grammar.

A parser is the module of an interpreter or compiler which performs parsing. It takes a sequence of tokens from the lexical analyzer finds a derivation for the sequence of tokens, and builds a parse tree (also known as a syntax tree) representing the derivation.

Two kinds of methods:
Top-down: constructs a parse tree from root to leaves
Bottom-up: constructs a parse tree from leaves to root

Recursive descent parsing is a top-down method of syntax analysis in which a set of recursive procedures is used to process the input. One procedure is associated with each nonterminal of a grammar.If a nonterminal has multiple productions, each production is implemented in a branch of a selection statement based on input lookahead information

Grammar without left recursion

E->TE'
E'->+TE'|e
T->FT'
T'->*FT'|e
F->(E)|i

Input expression: (i*i)+i

Stack Content	**Sequence of production rules**	**Expression**
E=$		(i*i)+i $
E=TE'$	E->TE'	(i*i)+i $
E=FT'E'$	T->FT'	(i*i)+i $
E=(E)T'E$'	F->(E)	i*i+i $
E=(TE')T'E$'	E->TE'	i*i+i $
E=(FT'E')T'E$'	T->FT'	i*i+i $
E=(iT'E')T'E' $	F->i	*i+i $
E=(i*FT'E')T'E'$	T'->*FT'	i+i $
E=(i*iT'E')T'E'$	F->i	+i $
E=(i*ieE')T'E' $	T'->e	+i $
E=(i*ie)T'E'$	E'->e	+i $
E=(i*i)eE'$	T'->e	+i $
E=(i*i)+TE'$	E'->+TE'	i $

Stack Content	**Sequence of production rules**	**Expression**
E=(i*i)+FT'E$'	T->FT'	i $
E=(i*i)+iT'E'$	F->i	$
E=(i*i)+ieE'$	T'->e	$
E=(i*i)+ie$	E'->e	$
E=(i*i)+i $		$

Program:

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
using namespace std;



char *ip=new char[100];
char *op=new char[100];
char *temp=new char[100];
int ip_ptr=0;
int n=0;

void e_dash();
void e();

void t_dash();

void t();

void f();

void advance();


void e()

{

    int n=0;

    for(int i=0;i<=strlen(op);i++) //remove epsilon

    {

           if(op[i]!='e')

                     temp[n++]=op[i];

    }

    strcpy(op,temp);

    for(n=0;n<strlen(op);n++)     //serching the nonterminal E

    {

           if(op[n]=='E')

                     break;

    }

    for(int i=n+1;i<=strlen(op);i++)

                     temp[i+2]=op[i];                     //  For  replacing  another  nonterminal

we moved some non terminals

          // For replacing non terminal

    temp[n]='T';
    temp[n+1]='E';
    temp[n+2]='\";
    strcpy(op,temp);
    printf("E=%-25s",op);
    printf("E->TE'\n");
    t();
    e_dash();
}

void e_dash()
{
    int n=0;
    for(int i=0;i<=strlen(op);i++)
    {
        if(op[i]!='e')
            temp[n++]=op[i];
    }
    strcpy(op,temp);
    for(n=0;n<strlen(op);n++)
    {
        if(op[n]=='E')
            break;
    }

    if(ip[ip_ptr]=='+')
```

```
{
    advance();
    strcpy(temp,op);
    for(int i=n+2;i<=strlen(op);i++)
        temp[i+2]=op[i];
    temp[n]='+';
    temp[n+1]='T';
    temp[n+2]='E';
    temp[n+3]='\"';
    strcpy(op,temp)
    ;
    printf("E=%-25s",op);
    printf("E'->+TE'\n");
    t();
    e_dash();
}
else
{
    strcpy(temp,op);
    for(int i=n+2;i<=strlen(op);i++)
                    temp[i-1]=op[i];
    temp[n]='e';
    strcpy(op,temp);
    printf("E=%-25s",op);
    printf("E'->e\n");
    }
}

void t()
{
    int n=0;
    for(int i=0;i<=strlen(op);i++)
    {
        if(op[i]!='e')
            temp[n++]=op[i];
    }
    strcpy(op,temp);
    for(n=0;n<strlen(op);n++)
    {
        if(op[n]=='T')
            break;
    }

    for(int i=n+1;i<=strlen(op);i++)
        temp[i+2]=op[i];
    temp[n]='F';
    temp[n+1]='T';
    temp[n+2]='\";
    strcpy(op,temp);
    printf("E=%-25s",op);
    printf("T->FT'\n");
    f();
    t_dash();
}

void t_dash()
{
    int n=0;
    for(int i=0;i<=strlen(op);i++)
    {
        if(op[i]!='e')
            temp[n++]=op[i];
    }
    strcpy(op,temp);
    for(n=0;n<strlen(op);n++)
    {
        if(op[n]=='T')
            break;
    }

    if(ip[ip_ptr]=='*')
    {
        advance();
        strcpy(temp,op);
        for(int i=n+2;i<=strlen(op);i++)
```

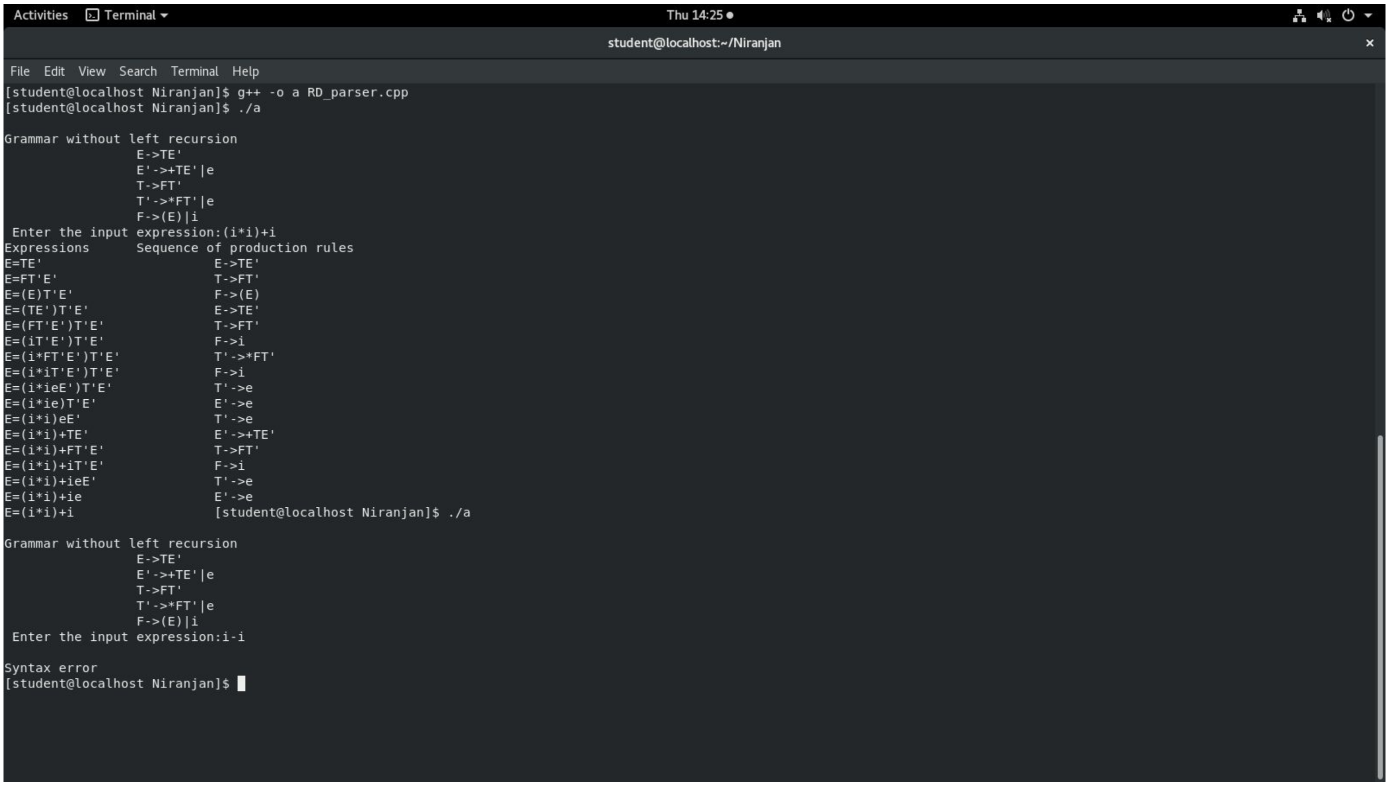
```
            temp[i+2]=op[i];
        temp[n]='*';
        temp[n+1]='F';
        temp[n+2]='T';
        temp[n+3]='\";
        strcpy(op,temp);
        printf("E=%-25s",op);
        printf("T'->*FT'\n");
        f();
        t_dash();
    }
    else
    {
        strcpy(temp,op);
        for(int i=n+2;i<=strlen(op);i++)
                        temp[i-1]=op[i];
        temp[n]='e';
        strcpy(op,temp);
        printf("E=%-25s",op);
        printf("T'->e\n");
    }
}

void f()
{
```

```
int n=0;
for(int i=0;i<=strlen(op);i++)
{
	if(op[i]!='e')
		temp[n++]=op[i];
}
strcpy(op,temp);
for(n=0;n<strlen(op);n++)
{
	if(op[n]=='F')
		break;
}

if(ip[ip_ptr]=='(')
{
	advance();
	strcpy(temp,op);
	for(int i=n+1;i<=strlen(op);i++)
		temp[i+2]=op[i];
	temp[n]='(';
	temp[n+1]='E';
	temp[n+2]=')';
	strcpy(op,temp);
	printf("E=%-25s",op);
	printf("F->(E)\n");
```

```
        e();
        if(ip[ip_ptr]==')')
        {
            advance();
        }
        else
        {
            printf("\n\t syntax error\n");
            exit(1);
        }
    }
    else if(ip[ip_ptr]=='i' || ip[ip_ptr]=='I')
    {
        advance();
        op[n]='i';
        printf("E=%-25s",op);
        printf("F->i\n");
    }
    else
    {
        printf("\n\t syntax error\n");
        exit(1);
    }
}
```

```
void advance()
{
    ip_ptr++;
}

int main()
{
 int i;
 int flag = 0;
 printf("\nGrammar without left recursion");
 printf("\n\t\t E->TE' \n\t\t E'->+TE'|e \n\t\t T->FT' ");
 printf("\n\t\t T'->*FT'|e \n\t\t F->(E)|i");
 printf("\n Enter the input expression:");
 scanf("%s",ip);
 for(i=0;i<strlen(ip);i++)
 {
  if(ip[i]!='+'&&ip[i]!='*'&&ip[i]!='('&&ip[i]!=')'&&ip[i]!='i'&&ip[i]!='I')
  {
   printf("\nSyntax error \n");

   flag = 1;
   break;
  }
 }
 if(flag == 0)
{
        printf("Expressions");
printf("\t Sequence of production rules\n");
strcpy(op,"");
e();

int n=0;
for(i=0;i<=strlen(op);i++)
{
   if(op[i]!='e')
         temp[n++]=op[i];
}

strcpy(op,temp);
printf("E=%-25s",op);
}

return 0;
}
```

Output:


Activities
Terminal
Thu 14:25
student@localhost:~/Niranjan
File Edit View Search Terminal Help

```
[student@localhost Niranjan]$ g++ -o a RD_parser.cpp
[student@localhost Niranjan]$ ./a

Grammar without left recursion
                 E->TE'
                 E'->+TE'|e
                 T->FT'
                 T'->*FT'|e
                 F->(E)|i
 Enter the input expression:(i*i)+i
Expressions      Sequence of production rules
E=TE'                      E->TE'
E=FT'E'                    T->FT'
E=(E)T'E'                  F->(E)
E=(TE')T'E'                E->TE'
E=(FT'E')T'E'              T->FT'
E=(iT'E')T'E'              F->i
E=(i*FT'E')T'E'            T'->*FT'
E=(i*iT'E')T'E'            F->i
E=(i*ieE')T'E'             T'->e
E=(i*ie)T'E'               E'->e
E=(i*i)eE'                 T'->e
E=(i*i)+TE'                E'->+TE'
E=(i*i)+FT'E'              T->FT'
E=(i*i)+iT'E'              F->i
E=(i*i)+ieE'               T'->e
E=(i*i)+ie                 E'->e
E=(i*i)+i                  [student@localhost Niranjan]$ ./a

Grammar without left recursion
                 E->TE'
                 E'->+TE'|e
                 T->FT'
                 T'->*FT'|e
                 F->(E)|i
 Enter the input expression:i-i

Syntax error
[student@localhost Niranjan]$ 
```

Conclusion:

Hene we have implemented recursive descent parser and check it for valid and invalid input strings.