

Department of Computer Science and Engineering
Compiler Design Lab (CS 306)

Week 6: Implementation of Recursive Descent Parser

Name : Pulkit Jasti | AP19110010491

Week 6 Programs

1. Implement Recursive Descent Parser for the Expression Grammar given below.

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow (E) \mid i \end{aligned}$$

Program:

C implementation of Recursive Descent Parser for the Expression Grammar is given below.

```
#include<stdio.h>
#include<string.h>
int E(),Edash(),T(),Tdash(),F();
char *ip;
char string[50];
int main()
{
    printf("Enter the string\n");
    scanf("%s",string);
    ip=string;
    printf("\n\nInput\tAction\n-----\n");
    if(E() && ip=='\0'){
        printf("\n-----\n");
        printf("\n String is successfully parsed\n"); }
    else{
        printf("\n-----\n");
        printf("Error in parsing String\n"); }
}
int E()
{
    printf("%s\tE->TE' \n",ip);
    if(T())
    {
        if(Edash())
```

```

{
return 1;
}
else
return 0;
}
else
return 0;
}
int Edash()
{
if(*ip=='+')
{
printf("%s\tE'->+TE' \n",ip);
ip++;
if(T())
{
if(Edash())
{
return 1;
}
else
return 0;
}
else
return 0;
}
else
{
printf("%s\tE'->^ \n",ip);
return 1;
}
}
}
int T()
{
printf("%s\tT->FT' \n",ip);
if(F())
{
if(Tdash())
{
return 1;
}
else
return 0;
}
else
return 0;
}
}
int Tdash()
{
if(*ip=='*')

```

```

{
printf("%s\tT'->*FT' \n",ip);
ip++;
if(F())
{
if(Tdash())
{
return 1;
}
else
return 0;
}
else
return 0;
}
else
{
printf("%s\tT'->^ \n",ip);
return 1;
}
}
int F()
{
if(*ip=='(')
{
printf("%s\tF->(E) \n",ip);
ip++;
if(E())
{
if(*ip==')')
{
ip++;
return 0;
}
else
return 0;
}
}
else if(*ip=='i')
{
ip++;
printf("%s\tF->id \n",ip);
return 1;
}
else
return 0;
}
}

```

Test cases:

i+i*i	String is successfully parsed
i+i	String is successfully parsed
i*i	String is successfully parsed
i*i+i*i+i	String is successfully parsed
i++i	Error in parsing String
i+i*	Error in parsing String

2. Construct Recursive Descent Parser for the grammar

$G = (\{S, L\}, \{(,), a, ,\}, \{S \rightarrow (L) \mid a ; L \rightarrow L, S \mid S\}, S)$ and verify the acceptability of the following strings:

- i. (a,(a,a))
- ii. (a,((a,a),(a,a)))

You can manually eliminate Left Recursion if any in the grammar.

Grammar :

$S \rightarrow (L) \mid a$

$L \rightarrow SL' \mid ,SL'$

$L' \rightarrow \wedge$

Program :

```
#include<stdio.h>
```

```
#include<string.h>
```

```

int S(),Ldash(),L();
char *ip;
char string[50];
int main()
{
    printf("Enter the string\n");
    scanf("%s",string);
    ip=string;
    printf("\n\nInput\t\tAction\n");
    if(S() && *ip=='\0')
    {
        printf("\n String is successfully parsed\n");
    }
    else
    {
        printf("Error in parsing String\n");
    }
}
int S()
{
    if(*ip=='(')
    {
        printf("%s\t\tS->(L) \n",ip);
        ip++;
        if(L())
        {
            if(*ip==')')
            {
                ip++;
                return 1;
            }

```

```

else
{
    return 0;
}
}
else
{
    return 0;
}
}
else if(*ip=='a')
{
    ip++;
    printf("%s\t\tS->a \n",ip);
    return 1;
}
else
{
    return 0;
}
}
int L()
{
    printf("%s\t\tL->SL' \n",ip);
    if(S())
    {
        if(Ldash())
        {
            return 1;
        }
    }
    else

```

```

    {
    return 0;
    }
    }
    else
    {
    return 0;
    }
}
int Ldash()
{
    if(*ip=='')
    {
    printf("%s\t\tL'->,SL' \n",ip);
    ip++;
    if(S())
    {
    if(Ldash())
    {
        return 1;
    }
    else
    {
        return 0;
    }
    }
    else
    {
    return 0;
    }
    }
}

```

```

else
{
printf("%s\t\tL'->^\n",ip);
return 1;
}
}

```

Test Cases :

```

Enter the string
(a,(a,a))

Input          Action
(a,(a,a))      S->(L)
a,(a,a))      L->SL'
,(a,a))      S->a
,(a,a))      L'->,SL'
(a,a))      S->(L)
a,a))      L->SL'
,a))      S->a
,a))      L'->,SL'
))      S->a
))      L'->^
)      L'->^

String is successfully parsed

```


Input	Action
(a, ((a,a), (a,a)))	S->(L)
a, ((a,a), (a,a)))	L->SL'
, ((a,a), (a,a)))	S->a
, ((a,a), (a,a)))	L' ->, SL'
((a,a), (a,a)))	S->(L)
(a,a), (a,a)))	L->SL'
(a,a), (a,a)))	S->(L)
a,a), (a,a)))	L->SL'
,a), (a,a)))	S->a
,a), (a,a)))	L' ->, SL'
), (a,a)))	S->a
), (a,a)))	L' ->^
, (a,a)))	L' ->, SL'
(a,a)))	S->(L)
a,a)))	L->SL'
,a)))	S->a
,a)))	L' ->, SL'
)))	S->a
)))	L' ->^
))	L' ->^
)	L' ->^

String is successfully parsed

Enter the string

(a,a))

Input	Action
(a,a))	$S \rightarrow (L)$
a,a))	$L \rightarrow SL'$
,a))	$S \rightarrow a$
,a))	$L' \rightarrow ,SL'$
)	$S \rightarrow a$
)	$L' \rightarrow ^$

Error in parsing String