

Week 7: Implementation of LL(1) parser using C

Nikitha Godavarthi

AP19110010460

CSE - C

Week 7 Program

1. Implement a non-recursive Predictive Parser for the grammar

$S \rightarrow aBa$

$B \rightarrow bB \mid \epsilon$

	a	b	\$
S	$S \rightarrow aBa$		
B	$B \rightarrow \epsilon$	$B \rightarrow bB$	

Programs:

Code of first program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
int i=0,top=0;
char stack[20],ip[20];

void push(char c)
{
    if (top>=20)
        printf("Stack Overflow");
    else
        stack[top++]=c;
}

void pop(void)
{
    if(top<0)
        printf("Stack underflow");
    else
        top--;
}

void error(void)
{
    printf("\n\nSyntax Error!!!! String is invalid\n");
    exit(0);
}
```

```

int main()
{
int n;

printf("The given grammar is\n\n");
printf("S -> aBa\n");
printf("B -> bB | epsilon \n\n");
printf("Enter the string to be parsed:\n");
scanf("%s",ip);
n=strlen(ip);
ip[n]='$';
ip[n+1]='\0';
push('$');
push('S');
while(ip[i]!='\0')
{ if(ip[i]=='$' && stack[top-1]=='$')
{
printf("\n\n Successful parsing of string \n");
return 1;
}
else
if(ip[i]==stack[top-1])
{
printf("\nmatch of %c ",ip[i]);
i++;pop();
}
else
{
if(stack[top-1]=='S' && ip[i]=='a')
{
printf(" \n S ->aBa");
pop();
push('a');
push('B');
push('a');
}
else
if(stack[top-1]=='B' && ip[i]=='b')
{
printf("\n B ->bB");
pop();push('B');push('b');
}
else
if(stack[top-1]=='B' && ip[i]=='a')
{
printf("\n B -> epsilon");
pop();
}
else
error();
}
}
}

```

```
    }  
  }  
} //end of main  
  
}
```

OUTPUT:

```
The given grammar is  
  
S -> aBa  
B -> bB | epsilon  
  
Enter the string to be parsed:  
abBa  
  
  S ->aBa  
match of a occurred  
  B ->bB  
match of b occurred  
match of B occurred  
match of a occurred  
  
  Successful parsing of string  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

```
The given grammar is  
  
S -> aBa  
B -> bB | epsilon  
  
Enter the string to be parsed:  
aaaa  
  
  S ->aBa  
match of a occurred  
  B -> epsilon  
match of a occurred  
  
Syntax Error!!!! String is invalid  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

2. Lab Assignment: Implement Predictive Parser using C for the Expression Grammar

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

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
int i=0,top=0;
char stack[20],ip[20];

void push(char c)
{
    if (top>=20)
        printf("Stack Overflow");
    else
        stack[top++]=c;
}

void pop(void)
{
    if(top<0)
        printf("Stack underflow");
    else
        top--;
}

void error(void)
{
    printf("\n\nSyntax Error!!! String is invalid\n");
    getch();
    exit(0);
}

int main()
{
    int n;

    printf("The given grammar is\n\n");
    printf("E -> TC\n");
    printf("C -> +TC | epsilon\n");
    printf("T -> FD\n");
    printf("D -> *FD | epsilon\n");
    printf("F -> (E) | d\n\n");
    printf("Enter the string to be parsed:\n");
```

```

scanf("%s",ip);
n=strlen(ip);
ip[n]='$';
ip[n+1]='\0';
push('$');
push('E');
printf("\ninput\t\taction\n");
while(ip[i]!='\0')
{
    if(ip[i]=='$' && stack[top-1]=='$')
    {
        printf("\n\n Successful parsing of string \n");
        return(1);
    }
    else if(ip[i]==stack[top-1])
    {
        printf("match of %c occurred ",ip[i]);
        i++;
        pop();
    }
    else
    {
        if(stack[top-1]=='E' && ip[i]=='d')
        {
            printf("\nE ->TC\t\t");
            pop();
            push('C');
            push('T');
        }
        else if(stack[top-1]=='E' && ip[i]=='(')
        {
            printf("\nE ->TC\t\t");
            pop();
            push('C');
            push('T');
        }
        else if(stack[top-1]=='C' && ip[i]=='+')
        {
            printf("\nC -> +TC\t\t");
            pop();
            push('C');
            push('T');
            push('+');
        }
        else if(stack[top-1]=='C' && ip[i]==')')
        {
            printf("\nC -> epsilon\t\t");
            pop();
        }
        else if(stack[top-1]=='C' && ip[i]=='$')

```

```

{
    printf("\nC -> epsilon\t");
    pop();
}
else if(stack[top-1]=='T' && ip[i]=='d')
{
    printf("\nT -> FD\t\t");
    pop();
    push('D');
    push('F');
}
else if(stack[top-1]=='T' && ip[i]=='(')
{
    printf("\nT -> FD\t\t");
    pop();
    push('D');
    push('F');
}
else if(stack[top-1]=='D' && ip[i]=='+')
{
    printf("\nD -> epsilon\t");
    pop();
}
else if(stack[top-1]=='D' && ip[i]=='*')
{
    printf("\nD -> *FD\t");
    pop();
    push('D');
    push('F');
    push('*');
}
else if(stack[top-1]=='D' && ip[i]==')')
{
    printf("\nD -> epsilon\t");
    pop();
}
else if(stack[top-1]=='D' && ip[i]=='$')
{
    printf("\nD -> epsilon\t");
    pop();
}
else if(stack[top-1]=='F' && ip[i]=='d')
{
    printf("\nF -> d\t\t");
    pop();
    push('d');
}
else if(stack[top-1]=='F' && ip[i]=='(')
{
    printf("\nF -> (E)\t");
    pop();
}

```

```

        push('');
        push('E');
        push('(');
    }
    else
    {
        error();
    }
}
}
}

```

OUTPUT:

```

The given grammar is
E -> TC
C -> +TC | epsilon
T -> FD
D -> *FD | epsilon
F -> (E) | d

Enter the string to be parsed:
d+d*d

input          action
E ->TC
T ->FD
F -> d          match of d occurred
D -> epsilon
C -> +TC        match of + occurred
T ->FD
F -> d          match of d occurred
D -> *FD        match of * occurred
F -> d          match of d occurred
D -> epsilon
C -> epsilon

Successful parsing of string

...Program finished with exit code 0
Press ENTER to exit console.

```

```

The given grammar is
E -> TC
C -> +TC | epsilon
T -> FD
D -> *FD | epsilon
F -> (E) | d

Enter the string to be parsed:
d+d*D

input          action
E ->TC
T ->FD
F -> d          match of d occurred
D -> epsilon
C -> +TC        match of + occurred
T ->FD
F -> d          match of d occurred
D -> *FD        match of * occurred

Syntax Error!!! String is invalid

...Program finished with exit code 0
Press ENTER to exit console.

```

The given grammar is

```
E -> TC
C -> +TC | epsilon
T -> FD
D -> *FD | epsilon
F -> (E) | d
```

Enter the string to be parsed:

d+d*d+d

input	action
E ->TC	
T ->FD	
F -> d	match of d occurred
D -> epsilon	
C -> +TC	match of + occurred
T ->FD	
F -> d	match of d occurred
D -> *FD	match of * occurred
F -> d	match of d occurred
D -> epsilon	
C -> +TC	match of + occurred
T ->FD	
F -> d	match of d occurred
D -> epsilon	
C -> epsilon	

Successful parsing of string