G LAKSHMI SAI BHARGAVI
AP19110010405 - CSE - C

CD LAB ASSIGNMENT - WEEK 7

1. Implement non-recursive Predictive Parser for the grammar

S -> aBa
B -> bB | ε

	a	b	$
S	S→aBa		
B	B→ε	B→b B	

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
}
```

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

E → TE'
E'→ +TE' | ε
T → FT'
T'→ *FT' | ε
F → (E) | d

The grammar is renamed for convenience the renamed grammar is as follows:
E → TA
A → +TA | ε
T → FB
B → *FB | ε
F → (E) | d

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("E -> TA\n");
printf("A -> +TA | epsilon \n\n" );
printf("T -> FB\n");
printf("B -> *FB | epsilon \n");
printf("F -> (E) | d\n");
printf("Enter the string to be parsed:\n");
scanf("%s",ip);
n=strlen(ip);
ip[n]='$';
ip[n+1]='\0';
push('$');
push('E');
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]=='E' && ip[i]=='d') || (stack[top-1]=='E' && ip[i]=='('))
{
```

```
printf(" \n E ->TA");
pop();
push('A');
push('T');

}
else
if(stack[top-1]=='A' && ip[i]=='+')

{
printf("\n A -> +TA");
pop();push('A');push('T');push('+');
}
else
if((stack[top-1]=='A' && ip[i]==')') || (stack[top-1]=='A' && ip[i]=='$'))
{
printf("\n A -> epsilon");
pop();
}
else
        if((stack[top-1]=='T' && ip[i]=='d') || (stack[top-1]=='T' && ip[i]=='(') )
        {
                printf("T -> FB\n");
                pop();
                push('B');
    push('F');
        }
else
        if(stack[top-1]=='B' && ip[i]=='*' )
        {
                printf("B -> *FB\n");
                pop();
                push('B');
    push('F');
    push('*');
        }
else
if((stack[top-1]=='B' && ip[i]=='+') ||  (stack[top-1]=='B' && ip[i]=='$') || (stack[top-1]=='B' &&
ip[i]==')'))
{
printf("\n B -> epsilon");
```

```
pop();
}
else
        if(stack[top-1]=='F' && ip[i]=='d' )
        {
                printf("F -> d\n");
                pop();
                push('d');

        }

else
        if(stack[top-1]=='F' && ip[i]=='(' )
        {
                printf("F -> (E)\n");
                pop();
                push(')');
                push('E');
                push('(');

        }

else
error();

}
}
```

Test cases:

```
The given grammar is

E -> TA
A -> +TA | epsilon

T -> FB
B -> *FB | epsilon
F -> (E) | d
Enter the string to be parsed:
d+d

 E ->TAT -> FB
F -> d

match of d
 B -> epsilon
 A -> +TA
match of + T -> FB
F -> d

match of d
 B -> epsilon
 A -> epsilon

 Successful parsing of string
```

```
The given grammar is

E -> TA
A -> +TA | epsilon

T -> FB
B -> *FB | epsilon
F -> (E) | d
Enter the string to be parsed:
d*d

 E ->TAT -> FB
F -> d

match of d B -> *FB

match of * F -> d

match of d
 B -> epsilon
 A -> epsilon

 Successful parsing of string
```

```
E -> TA
A -> +TA | epsilon

T -> FB
B -> *FB | epsilon
F -> (E) | d
Enter the string to be parsed:
d*d+d

 E ->TAT -> FB
F -> d

match of d B -> *FB

match of * F -> d

match of d
 B -> epsilon
 A -> +TA
match of + T -> FB
F -> d

match of d
 B -> epsilon
 A -> epsilon

 Successful parsing of string
```

```
T -> FB
B -> *FB | epsilon
F -> (E) | d
Enter the string to be parsed:
(d+d)

 E ->TAT -> FB
F -> (E)

match of (
 E ->TAT -> FB
F -> d

match of d
 B -> epsilon
 A -> +TA
match of + T -> FB
F -> d

match of d
 B -> epsilon
 A -> epsilon
match of )
 B -> epsilon
 A -> epsilon

 Successful parsing of string


Press any key to continue . . .
```