PART B
EXPERIMENT NUMBER 4

Aim: Write a program to implement any parsing technique.

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded at the end of the practical)

Roll No. 50	**Name:** AMEY THAKUR
Class: Comps TE B	**Batch:** B3
Date of Experiment: 26/03/2021	**Date of Submission:** 26/03/2021
Grade:	

B.1 Software Code written by a student:

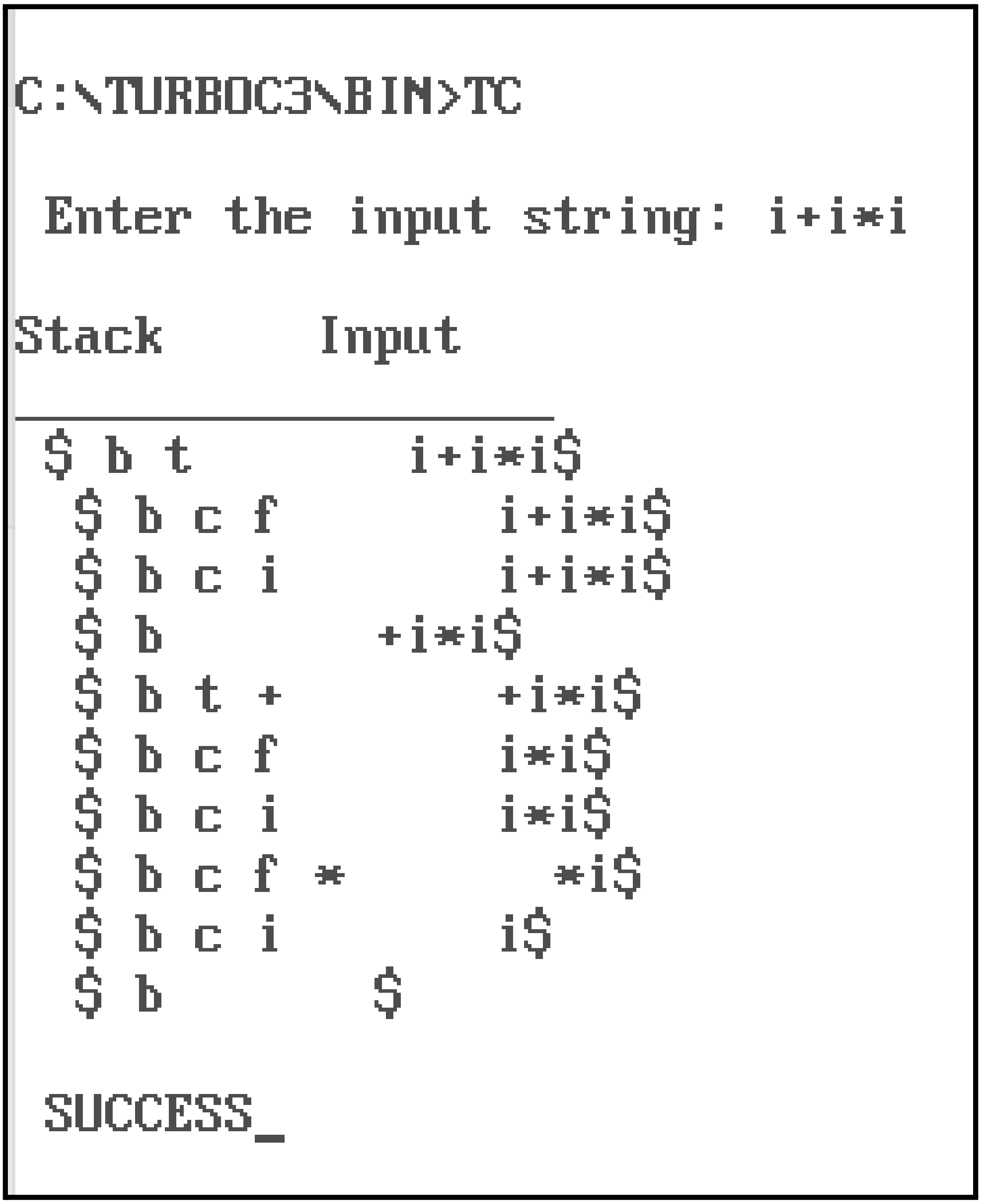
(Paste your code completed during the 2 hours of practice in the lab here)

- **SPCC-4.C**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s[20],stack[20];

int main()
{
char m[5][6][3] = {"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," ","
"," ","n","*fc"," a ","n","n","i"," "," ","(e)"," "," "};

int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
int i,j,k,n,str1,str2;

printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
j=0;
```

```
printf("\nStack     Input\n");
printf("___________________\n");

while((stack[i]!='$')&&(s[j]!='$'))
{
if(stack[i]==s[j])
{
i--;
j++;
}
switch(stack[i])
{
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'c': str1=3;
break;
case 'f': str1=4;
break;
}

switch(s[j])
{
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;
}

if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
getch();
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
i++;
}
i--;

}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf("       ");
for(k=j;k<=n;k++)
printf("%c",s[k]);
printf(" \n ");
}
printf("\n SUCCESS");
getch();
 }
```

B.2 Input and Output:



```
C:\TURBOC3\BIN>TC

 Enter the input string: i+i*i

Stack     Input
__________________
 $ b t       i+i*i$
  $ b c f       i+i*i$
  $ b c i       i+i*i$
  $ b       +i*i$
  $ b t +       +i*i$
  $ b c f       i*i$
  $ b c i       i*i$
  $ b c f *       *i$
  $ b c i       i$
  $ b       $

 SUCCESS_
```

B.3 Observations and learning:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ subpart assigned)

We have learnt about recursive grammar and how to remove the left recursion. Also, we learnt to deduce the first and follow set for each non-terminal and by using the first and follow set we have learnt predictive parsing.

B.4 Conclusion:

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

Thus we have studied and implemented the C program for predictive parsing.

B.5 Question of Curiosity

(To be answered by a student based on the practical performed and learning/ observations)

1. What is the mechanism of the Top-Down parser?

Ans:

A top-down parser starts with the root of the parse tree. The root node is labelled with the goal symbol of the grammar.

Top-down parsing algorithm:

1. Construct the root node of the parse tree.
2. Repeat until the leaves of the parse tree match the input string.
3. At a node labelled A, select a production with A on its LHS and, for each symbol on its RHS, construct the appropriate child.
4. When a terminal symbol is added to the fringe and it doesn't match the fringe, backtrack.
5. Find the next node to be expanded.

2. How do you recognize LL(1) grammar?

Ans:

- ➔ To check if a grammar is LL(1), we must make sure that
    1. The grammar is not ambiguous
    2. The grammar should not be left recursive
    3. The grammar should be deterministic.
- ➔ The idea is that if you construct the LL(1) parsing table, no cell should have more than one entry.

3. What are the key differences between recursive and non-recursive-descent parsers?

Ans:

Recursive Predictive Descent Parser	Non-Recursive Predictive Descent Parser
It is a technique that may or may not require a backtracking process.	It is a technique that does not require any kind of backtracking.
It uses procedures for every non-terminal entity to parse strings.	It finds out productions to use by replacing the input string.
It is a type of top-down parsing built from a set of mutually recursive procedures where each procedure implements one of the non-terminals of grammar.	It is a type of top-down approach, which is also a type of recursive parsing that does not use a technique of backtracking.
It contains several small functions, one for each non-terminal in grammar.	The predictive parser uses a look ahead pointer which points to the next input symbols to make it parser backtracking free, predictive parser puts some constraints on grammar.
It accepts all kinds of grammar.	It accepts only a class of grammar known as LL(k) grammar.