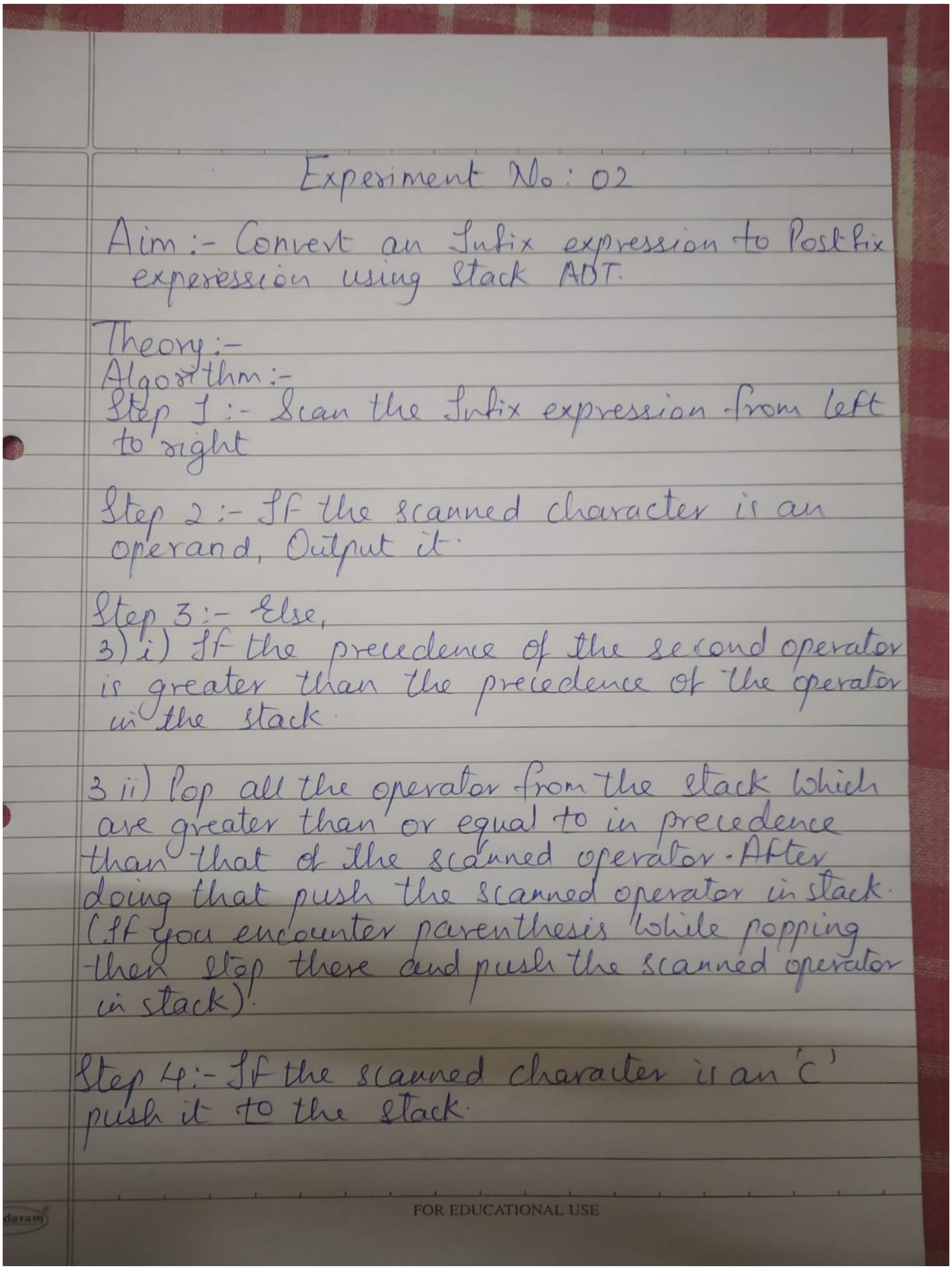
Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

COMPUTER ENGINEERING

DS ODD SEM 2021-22/EXPERIMENT 2

NAME:- GAURAV AMARNANI (D7A, 67)



Experiment No: 02

Aim:- Convert an Infix expression to Postfix experession using Stack ADT.

Theory:-
Algorithm:-
Step 1:- Scan the Infix expression from left to right

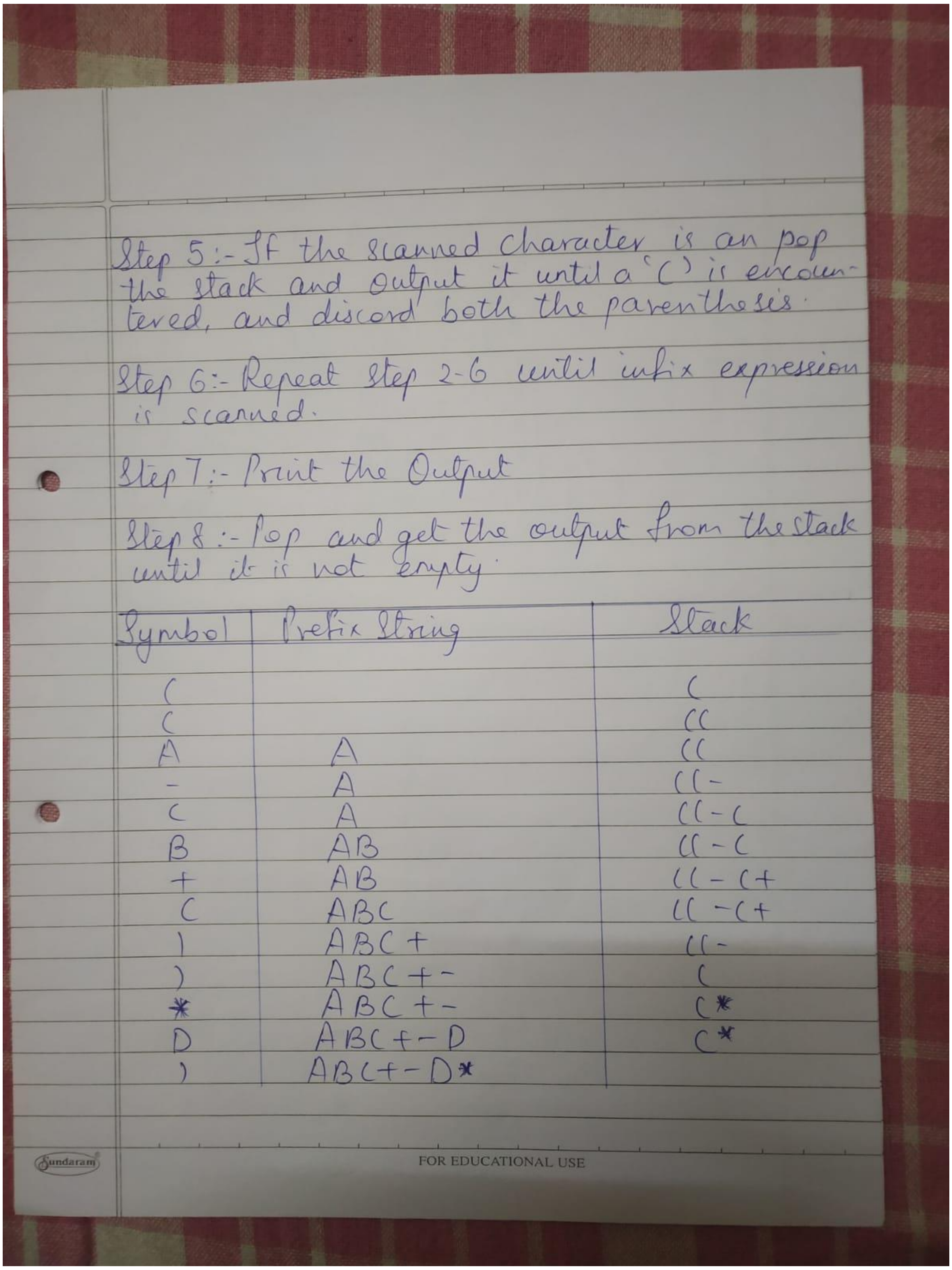
Step 2:- If the scanned character is an operand, Output it.

Step 3:- Else,
3) i) If the precedence of the second operator is greater than the precedence of the operator in the stack.

3 ii) Pop all the operator from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that push the scanned operator in stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in stack).

Step 4:- If the scanned character is an '('
push it to the stack.

FOR EDUCATIONAL USE



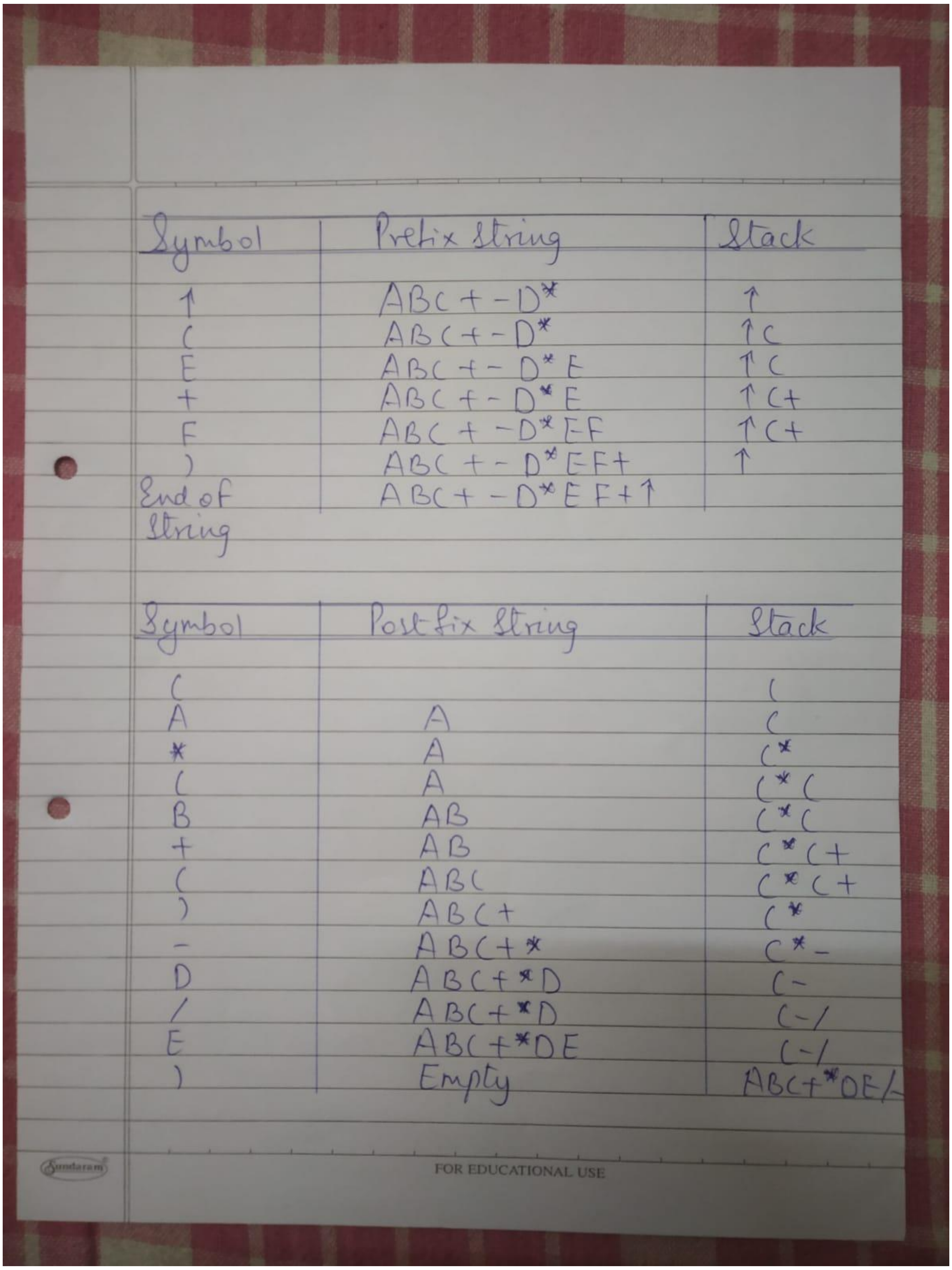
Step 5:- If the scanned character is an pop the stack and Output it until a '(' is encountered, and discord both the parenthesis.

Step 6:- Repeat step 2-6 until infix expression is scanned.

Step 7:- Print the Output

Step 8:- Pop and get the output from the stack until it is not empty.

Symbol	Prefix String	Stack
(		(
(		((
A	A	((
-	A	((-
(	A	((-(
B	AB	((-(
+	AB	((-(+
C	ABC	((-(+
)	ABC+	((-
)	ABC+-	(
*	ABC+-	(*
D	ABC+-D	(*
)	ABC+-D*	



Symbol	Prefix String	Stack
↑	ABC+-D*	↑
(	ABC+-D*	↑(
E	ABC+-D*E	↑(
+	ABC+-D*E	↑(+
F	ABC+-D*EF	↑(+
)	ABC+-D*EF+	↑
End of String	ABC+-D*EF+↑	

Symbol	Post fix String	Stack
(		(
A	A	(
*	A	(*
(	A	(*(
B	AB	(*(
+	AB	(*(+
(	ABC	(*(+
)	ABC+	(*
-	ABC+*	(*-
D	ABC+*D	(-
/	ABC+*D	(-/
E	ABC+*DE	(-/
)	Empty	ABC+*DE/-

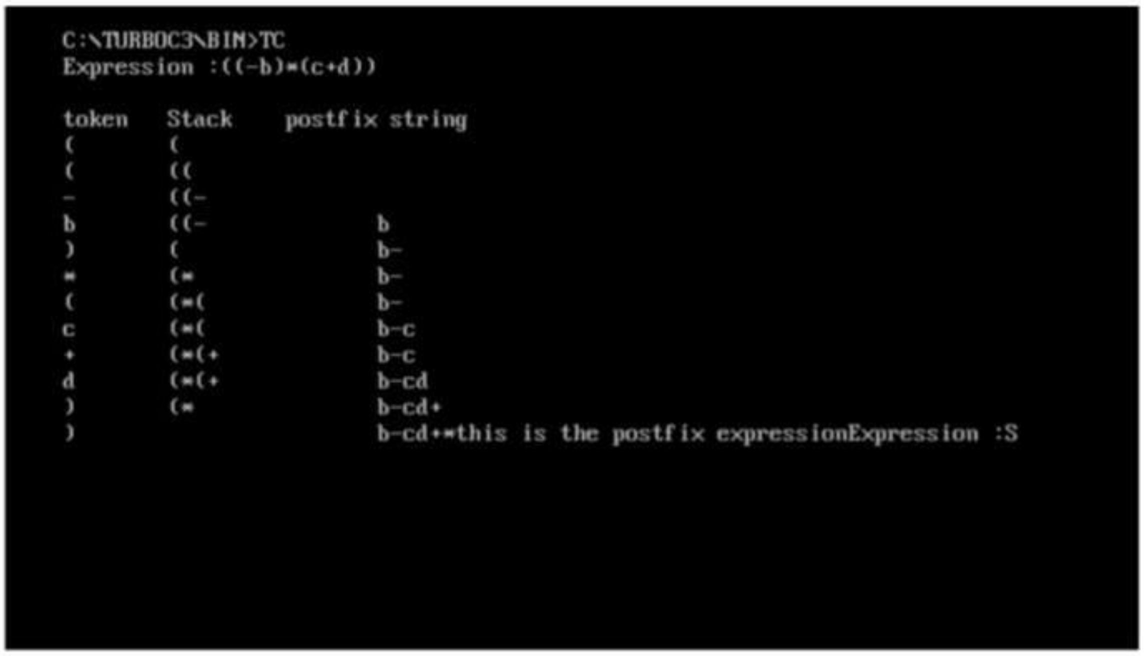
PROGRAM NO 1: WRITE A PROGRAM TO IMPLEMENT STACK USING LINKED LIST

```
#include<stdio.h>
#include<conio.h>
#define n 50
char stack[n];
int top=-1,j=0;
char postfix[50];
void push(char);
char pop();
int priority(char);
void main() {
int i;
char element,ch;
char infix[50];
printf("Expression:");
gets(infix);
printf("\ntoken\tStack \t postfix string");
for(i=0;infix[i]!=NULL;i++) {
ch=infix[i];
if(ch>='a' && ch<='z') {
postfix[j]=ch; j++;
}
else if(ch=='(') {
push(ch);
}
else if(ch==')') {
while((element=pop())!='(') {
postfix[j]=element; j++;
}
}
else {
while(priority(ch)<=priority(stack[top])) {
if(stack[top]=='(')
break;
element=pop();
postfix[j]=element;
j++;
}
push(ch);
}
postfix[j]=NULL;
printf("\n%c\t%s\t\t%s",ch,stack,postfix);
getch();
}
while((element=pop())!='(') {
postfix[j]=element;
j++;
}
getch();
}
void push(char ch) {
if(top>=n-1) {
```

```
printf("overflow");
}
else {
top=top+1;
stack[top]=ch;
}
}
char pop() {
char item;
if(top==- 1) {
printf("this is the postfix expression");
exit(0);
}
else {
top=top-1;
item=stack[top+1];
stack[top+1]=NULL;
}

return item;
}
int priority(char ch) {
char operand[6]={'+','-','*','/','(','\0'};
int prio[5]={1,1,2,2,3};
int i,a;
for(i=0;i<5;i++) {
if(ch==operand[i]) {
a=prio[i];
break;
}
}
return a;
}
```

<u>OUTPUT:-</u>



```
C:\TURBOC3\BIN>TC
Expression :((-b)*(c+d))

token   Stack    postfix string
(       (
(       ((
-       ((-
b       ((-            b
)       (              b-
*       (*             b-
(       (*(            b-
c       (*(            b-c
+       (*(+           b-c
d       (*(+           b-cd
)       (*             b-cd+
)                      b-cd+*this is the postfix expressionExpression :S
```