ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

COMPILER DESIGN LAB

EXP 6 - PREDICTIVE PARSING

AIM :

To find the first and follow from a given grammar and carry out predictive parsing

REQUIREMENTS :

1. Knowledge of the concepts of grammars and production rules
2. Knowledge of the concepts of First and Follow Algorithm
3. Knowledge of the concepts of predictive parsing and building parse tables
4. Online compiler for execution of C program

THEORY :

Predictive Parser :

A predictive parser is a recursive descent parser with no backtracking or backup. It is a top-down parser that does not require backtracking. At each step, the choice

of the rule to be expanded is made upon the next terminal symbol.
Consider
A -> A1 | A2 | ... | An
If the non-terminal is to be further expanded to ‘A’, the rule is selected based on the current input symbol ‘a’ only.

ALGORITHM :
The main Concept ->With the help of FIRST() and FOLLOW() sets, this parsing can be done using just a stack that avoids the recursive calls.
For each rule, A->x in grammar G:

1. For each terminal ‘a’ contained in FIRST(A) add A->x to M[A, a] in the parsing table if x derives ‘a’ as the first symbol.

2. If FIRST(A) contains null production for each terminal ‘b’ in FOLLOW(A), add this production (A->null) to M[A, b] in the parsing table.

PROCEDURE :
1. In the beginning, the pushdown stack holds the start symbol of the grammar G.

2. At each step a symbol X is popped from the stack: if X is a terminal then it is matched with the lookahead and lookahead is advanced one step,

If X is a nonterminal symbol, then using lookahead and a parsing table (implementing the FIRST sets) a production is chosen and its right-hand side is pushed into the stack.

3. This process repeats until the stack and the input string become null (empty).

SOURCE CODE :

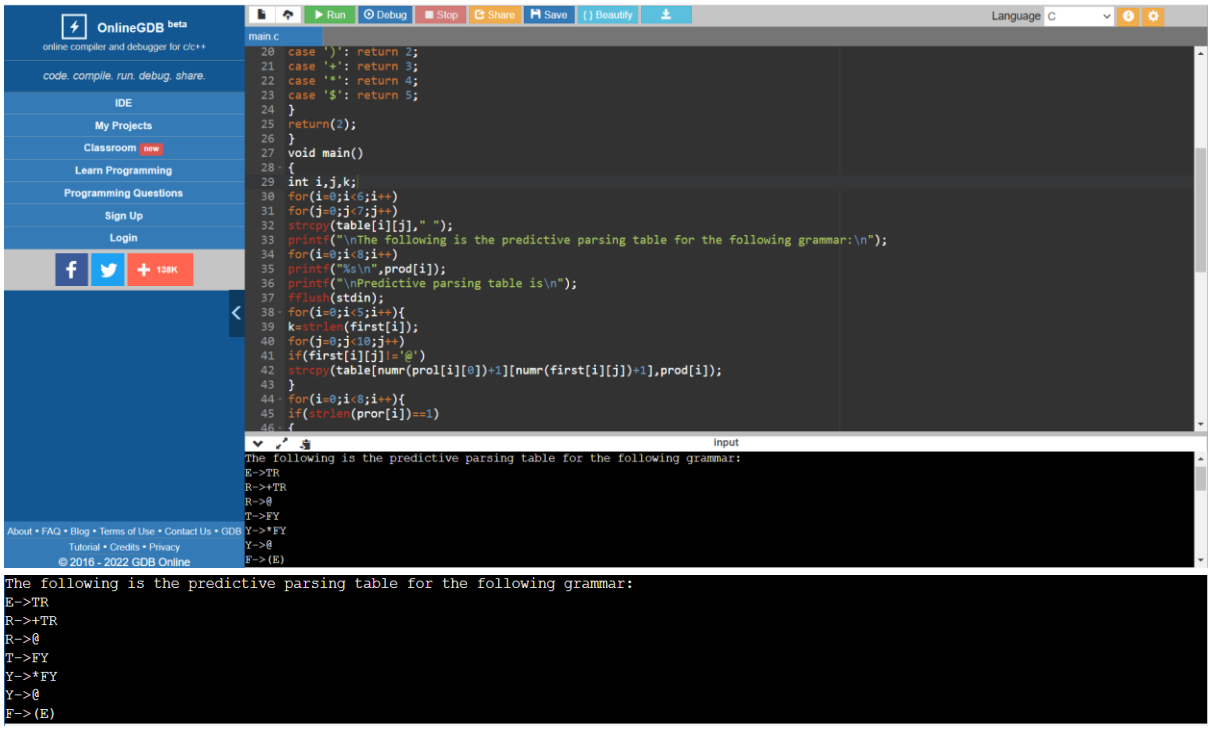
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char prol[8][10]={"E","R","R","T","Y","Y","F","F"};
char pror[8][10]={"TR","+TR","@","FY","*FY","@","(E)","i"};
char prod[8][10]={"E->TR","R->+TR","R->@","T->FY","Y->*FY","Y-
>@","F->(E)","F->i"};
char first[5][10]={"i(","+$","i(","*$","i("};
char follow[5][10]={"$)","$)","+$)","+$)","*$)+"};
char table[6][7][10];
int numr(char c)
{
switch(c){
case 'E': return 0;
case 'R': return 1;
case 'T': return 2;
case 'Y': return 3;
case 'F': return 4;
case 'i': return 0;
case '(': return 1;
case ')': return 2;
case '+': return 3;
case '*': return 4;
case '$': return 5;
```

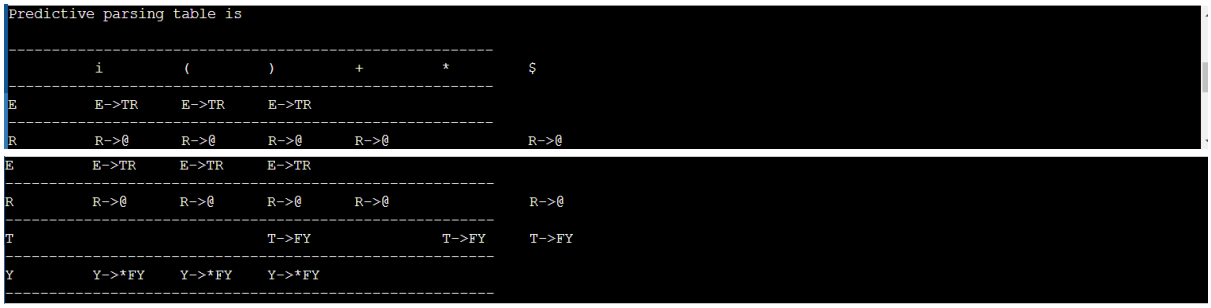
```
}
return(2);
}
void main()
{
int i,j,k;
clrscr();
for(i=0;i<6;i++)
for(j=0;j<7;j++)
strcpy(table[i][j]," ");
printf("\nThe following is the predictive parsing table for the following
grammar:\n");
for(i=0;i<8;i++)
printf("%s\n",prod[i]);
printf("\nPredictive parsing table is\n");
fflush(stdin);
for(i=0;i<5;i++){
k=strlen(first[i]);
for(j=0;j<10;j++)
if(first[i][j]!='@')
strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
}
for(i=0;i<8;i++){
if(strlen(pror[i])==1)
{
if(pror[i][0]=='@')
{
k=strlen(follow[i]);
for(j=0;j<k;j++)
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
}
}
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"i");
strcpy(table[0][2],"(");
strcpy(table[0][3],")");
```

```
strcpy(table[0][4],"+");
strcpy(table[0][5],"*");
strcpy(table[0][6],"$");
strcpy(table[1][0],"E");
strcpy(table[2][0],"R");
strcpy(table[3][0],"T");
strcpy(table[4][0],"Y");
strcpy(table[5][0],"F");

printf("\n----------------------------------------------------\n");
for(i=0;i<6;i++)
for(j=0;j<7;j++){
printf("%-10s",table[i][j]);
if(j==6)
printf("\n----------------------------------------------------\n");
}
getch();
}
```

SCREENSHOT OF OUTPUT :





OBSERVATION :

The first() and follow() of each non terminal symbols were found . In the stack , the start symbol was present and based on the given expression , substitutions were done until the stack and input string became empty.

RESULT :

Thus we have successfully carried out the process of predictive parsing.