INSTITUTE OF ENGINEERING & MANAGEMENT

AME OF THE STUDENT : Abhishek Anand.

EMESTER : 6th

OLL NO. : 57

SSIGNMENT OF EPARTMENT : compiler Design Lab

ATE OF EXPERIMENT / ROJECT : 19/04/2022

ATE OF SUBMISSION :

TLE : Predictive Parser

BJECT : C program to implement the functionalities of predictive parser for the mini language

Objective: Write a C program for implementing the functionalities of predictive parser for the mini language specified below.

S → A
A → Bb
A → Cd
B → aB
B → @
C → Cc
C → @

Resource: Online GDB, gcc/g++

Program logic & procedure :-

A predictive parser is a recursive descent parser with no backtracking or backup. The logic behind my code is -

(i) Read the input string
(ii) By using the first & follow values, verify the first of non-terminal & insert the production in follow values.
(iii) After implementing the previous step, it will construct the predictive parser table.

Program :

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char arr1[7][10] = {"S", "A", "A", "B", "B", "C", "C"};
char arr2[7][10] = {"A", "Bb", "Cd", "aB", "@", "Cc", "@"};
char arr3[7][10] = {"S->A", "A->Bb", "A->Cd", "B->aB", "B->@",
                    "C->Cc", "C->@"};
```

Roll No.: 57

Name: Abhishek Anand

Year: 3rd.

```
char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};
char follow[7][10]={"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
int abhi(char c){
    switch(c){
        case 's':
            return 0;
        case 'A':
            return 1;
        case 'B':
            return 2;
        case 'c':
            return 3;
        case 'a':
            return 0;
        case 'b':
            return 1;
        case 'c':
            return 2;
        case 'd':
            return 3;
        case '$':
            return 4;
    }
    return(2);
}
void main(){
    int i,j,k;
    printf("\n Predictive parsing table \n");
    fflush(stdin);
    for(i=0;i<7;i++){
        k=strlen(first[i]);
        for(j=0;j<10;j++)
        if(first[i][j]!='@')
```

Name:

Roll No.:

Year:

```
strcpy (table[abhi(arr1[i][0])+1][abhi(first[i][j]+1], arr3[i]);
    }
for(i=0; i<7; i++){
        if (strlen(arr2[i] == 1) {
            if (arr2[i][0] == '@') {
            k = strlen (follow[i]);
            for (j=0; j<k; j++)
strcpy (table[abhi(arr1[i][0])+1][abhi(follow[i][j])+1], arr3[i]);
        }
    }
}

strcpy (table[0][0], " ");
strcpy (table[0][1], "a");
strcpy (table[0][2], "b");
strcpy (table[0][3], "c");
strcpy (table[0][4], "d");
strcpy (table[0][5], "$");
strcpy (table[1][0]; "S");
strcpy (table[2][0], "A");
strcpy (table[3][0], "B");
strcpy (table[4][0], "C");

for (i=0; i<5; i++)
        for(j=0; j<6; j++){
    printf ("%-10s", table[i][j]);
    if (j==5)
        printf("\n ---------- \n");
                                // For seperator
    }
}
```

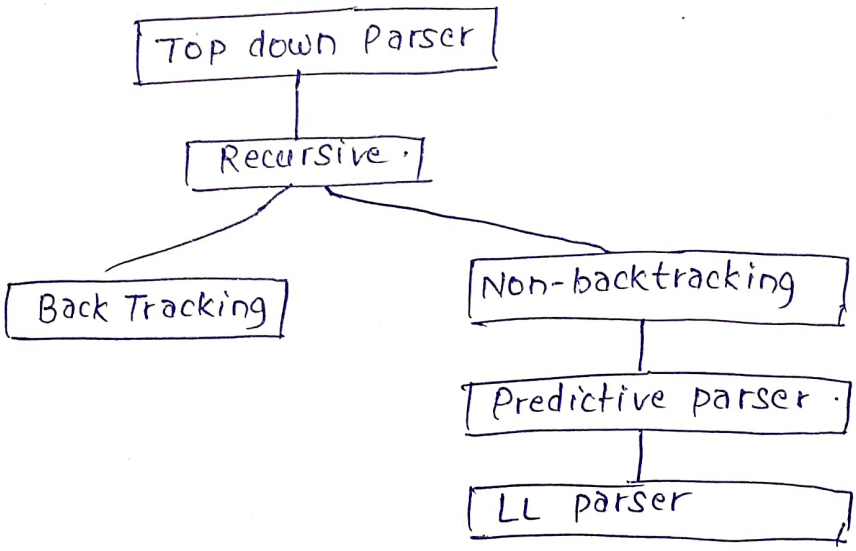
Output:

Predictive parsing table.

	a	b	c	d	$
S	S→A	S→A	S→A	S→A	
A	A→Bb	A→Bb	A→cd	A→Cd	
B	B→aB	B→@	B→@		B→@
C			C→@	c→@	c→@

Discussion:

As we know that in a predictive parser there will be no backtracking or backup. It doesnot require backtracking. At each step, the choice of the rule to be expanded is made upon the next terminal symbol.



So, The approach to solve this problem is by using the first & follow values. First of all, We verify the first of non-terminals & insert the Production in the FIRST value. If we have any @ value in FIRST then insert the productions in FOLLOW values.