Name – Bhargav Shamuvel Gurav | **PRN** – 2041009

Class – L.Y. B-Tech (Computer) | **Batch** – B1

Course Code – CO406U | **Course Name** - CDL

Practical no. 8

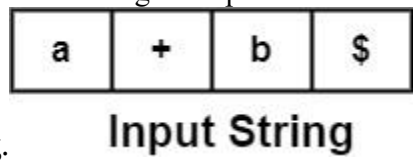
Aim: Design of a Predictive parser of a given language.

Theory :

Predictive Parser is also another method that implements the technique of Top-Down parsing without Backtracking. A predictive parser is an effective technique of executing recursive-descent parsing by managing the stack of activation records, particularly.

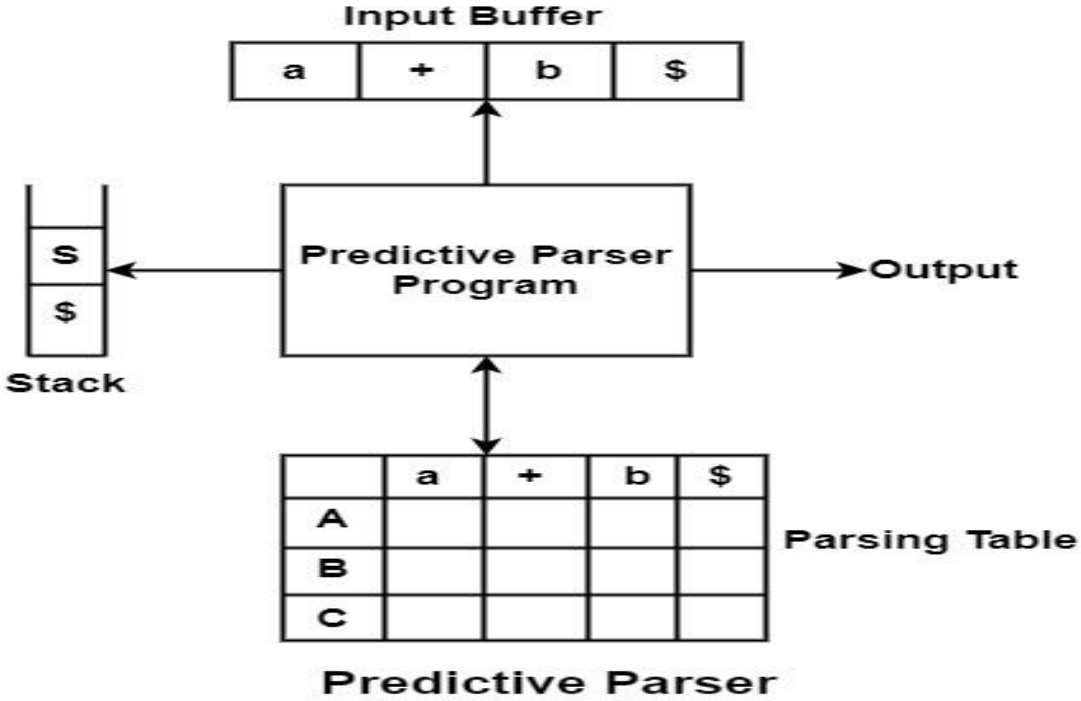
Predictive Parsers has the following components –

- **Input Buffer** − The input buffer includes the string to be parsed followed by an end marker $ to denote the end of the string.



Here a, +, b are terminal symbols.

Stack − It contains a combination of grammar symbols with $ on the bottom of the stack. At the start of Parsing, the stack contains the start symbol of Grammar followed by $.



- **Parsing Table** − It is a two-dimensional array or Matrix M [A, a] where A is nonterminal and 'a' is a terminal symbol.

All the terminals are written column-wise, and all the Non-terminals are written rowwise.

- **Parsing Program** − The parsing program performs some action by comparing the symbol on top of the stack and the current input symbol to be read on the input buffer.
- **Actions** − Parsing program takes various actions depending upon the symbol on the top of the stack and the current input symbol. Various Actions taken are given below −

Description	Top of Stack	Current Input Symbol	Action
1. If stack is empty, i.e., it only contains $ and current Input symbol is also $.	$	$	Parsing will be successful and will be halted.
2. If symbol at top of stack and the current input symbol to be read are both terminals and are same.	ⓐ $	ⓐ b $	Pop a from stack & advance to next input symbol.
3. If both top of stack & current input symbol are terminals and top of stack ≠ current input symbol e.g. a ≠ b.	ⓐ $	ⓑ $	Error
4. If top of stack is non-terminal & input symbol is terminal.	Ⓧ $	ⓐ $	Refer to entry M [X, a] in Parsing Table. If M[X, a] = X → ABC then Pop X from Stack Push C, B, A onto stack.

Algorithm to construct Predictive Parsing Table

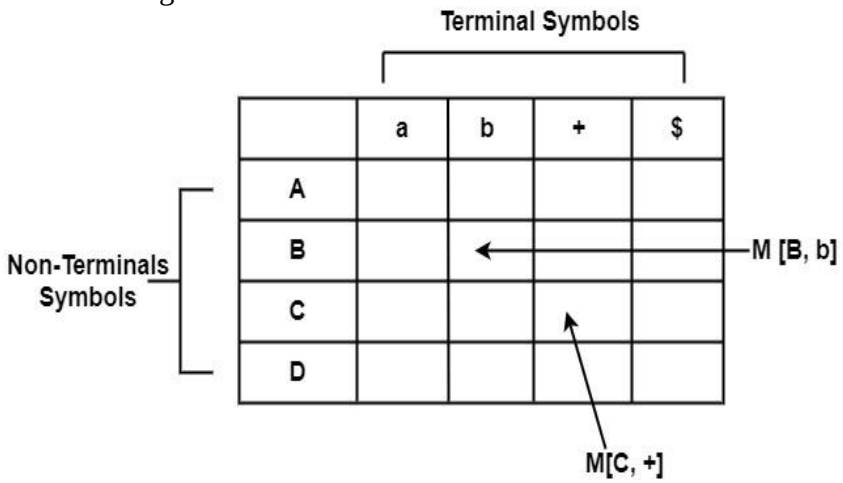
Input − Context-Free Grammar G

Output − Predictive Parsing Table M

Method − For the production A → ɑ of Grammar G.

- For each terminal, a in FIRST (α) add A → α to M [A, a].
- If ε is in FIRST (α), and b is in FOLLOW (A), then add A → α to M[A, b].
- If ε is in FIRST (α), and $ is in FOLLOW (A), then add A → α to M[A, $].

- All remaining entries in Table M are errors.



Following are the steps to perform Predictive Parsing

- Elimination of Left Recursion
- Left Factoring
- Computation of FIRST & FOLLOW
- Construction of Predictive Parsing Table
- Parse the Input String

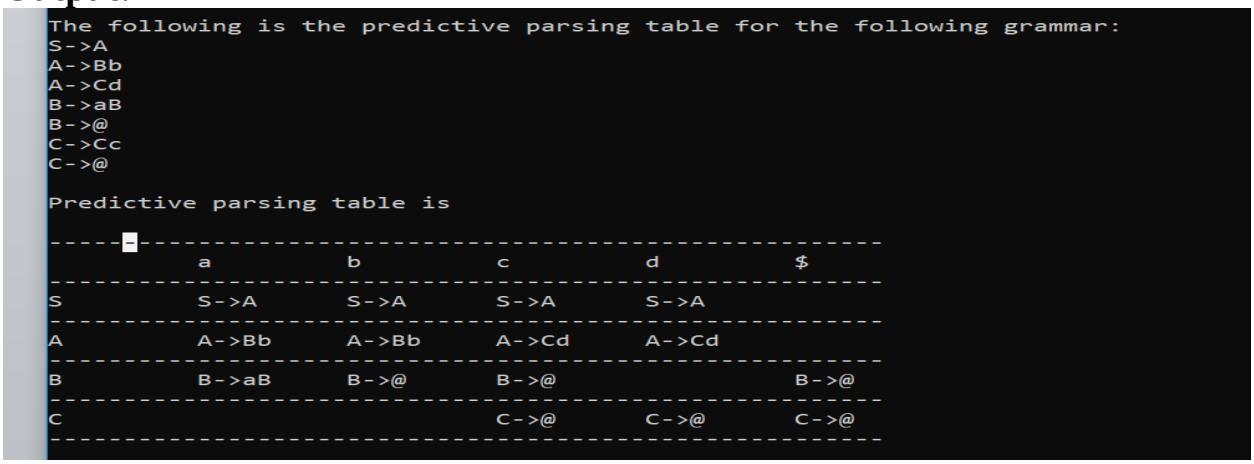
Program Code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char prol[7][10]={"S","A","A","B","B","C","C"};
char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C-
>@"};
char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};
char follow[7][10]={"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
numr(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;

for(i=0;i<5;i++)
for(j=0;j<6;j++)
strcpy(table[i][j]," ");
printf("\nThe following is the predictive parsing table for the following
grammar:\n");
for(i=0;i<7;i++)
printf("%s\n",prod[i]);
printf("\nPredictive parsing table is\n");
fflush(stdin);
for(i=0;i<7;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<7;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],"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");
printf("\n----------------------------------------------------\n");
for(i=0;i<5;i++)
for(j=0;j<6;j++)
{
printf("%-10s",table[i][j]);
if(j==5)
printf("\n----------------------------------------------------\n");
}
getch();
}
```

Output:



Conclusion : In this practical we implemented predictive parser.