1. Write a LEX Program to scan reserved word & Identifiers of C Language

```
%{
int n=0;
%}
%%
"if"|"else"|"while"|"for" { printf("\t keyword %s",yytext); }
[a-zA-Z_][a-zA-Z_]* { printf("\t identifier %s",yytext); }
"<"|"<="|">"|">="|"=="|"="|"+"|"-"|"*" { printf("\t operator %s",yytext);}
[(){},;] { printf("\t separator %s",yytext);}
[0-9]+ { printf("\t digits %s",yytext);}
%%
int main()
{
yylex();
}
int yywrap()
return 1;
}
```

Program Compilation and Execution

```
lex tok.l
gcc lex.yy.c
./a.out
c=a+b;
identifier c operator = identifier a operator + identifier b separator ;
^Z
```

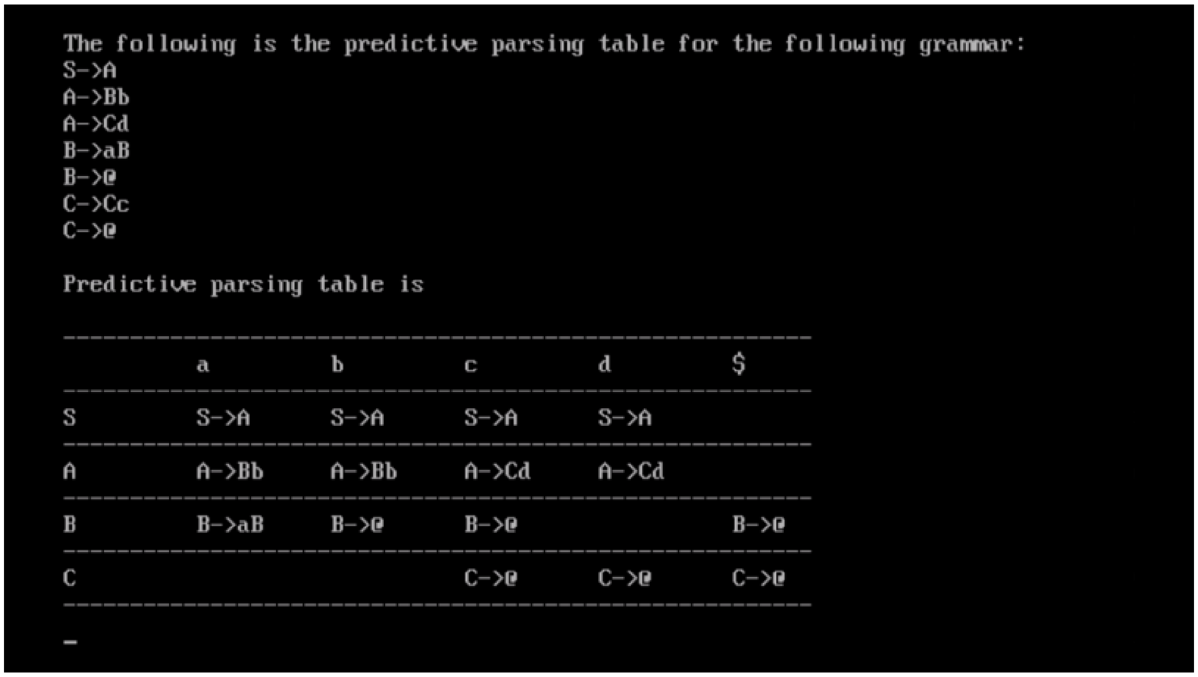
2. Implement Predictive Parsing algorithm using C program .

```
#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 '$': return 4;
}
return(2);
}
void main()
{
int i,j,k;
clrscr();
for(i=0;i<5;i++)
for(j=0;j<6;j++)
strcpy(table[i][j]," ");
printf("\nThe following is the predictive parsing table for the following grammar:\n");
for(i=0;i<7;i++)
printf("%s\n",prod[i]);
printf("\nPredictive parsing table is\n");
fflush(stdin);
for(i=0;i<7;i++)
{
k=strlen(first[i]);//first[1]=ab
for(j=0;j<k;j++)
if(first[i][j]!='@')//first[1][0]=b b!=@
strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);//table[2][1],prod[1]
}
for(i=0;i<7;i++)
{
if(strlen(pror[i])==1)
{
if(pror[i][0]=='@')
{
k=strlen(follow[i]);
for(j=0;j<k;j++)
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
}
}
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"d");
strcpy(table[0][5],"$");
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

OUTPUT



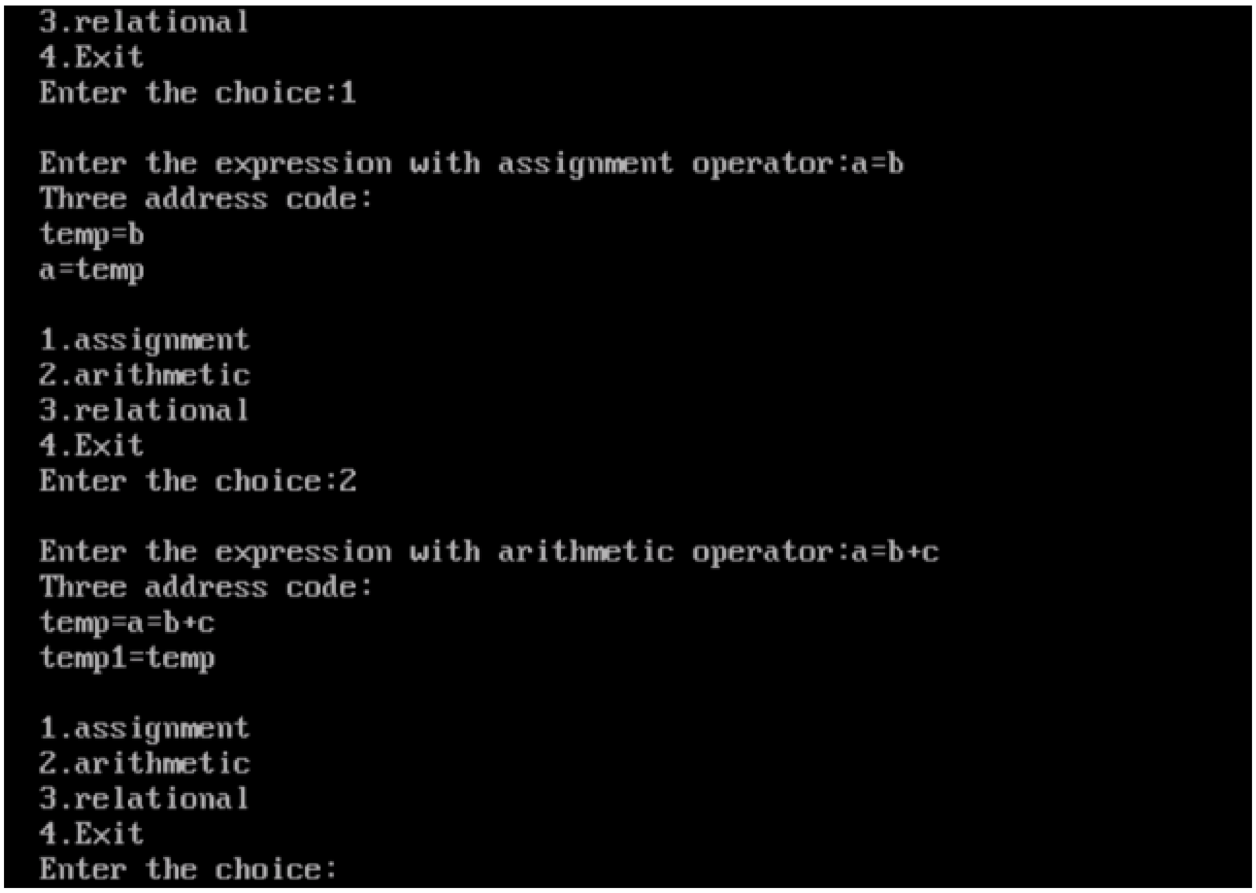
```
The following is the predictive parsing table for the following grammar:
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table is

--------------------------------------------------------------
          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->@
--------------------------------------------------------------
_
```

3. Write a C program to generate three address code

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
clrscr();
while(1)
{
printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nEnter the expression with assignment operator:");
scanf("%s",exp);
l=strlen(exp);
exp2[0]='\0';
i=0;
while(exp[i]!='=')
{
i++;
}
strncat(exp2,exp,i);
strrev(exp);
exp1[0]='\0';
strncat(exp1,exp,l-(i+1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
break;
case 2:
printf("\nEnter the expression with arithmetic operator:");
scanf("%s",ex);
strcpy(exp,ex);
l=strlen(exp);
exp1[0]='\0';
for(i=0;i<l;i++)
{
if(exp[i]=='+'||exp[i]=='-')
{
if(exp[i+2]=='/'||exp[i+2]=='*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if(exp[i]=='/'||exp[i]=='*')
{
div();
break;
}
}
break;
case 3:
printf("Enter the expression with relational operator");
scanf("%s%s%s",&id1,&op,&id2);
if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,">=")==0)||(strcm
p
(op,"==")==0)||(strcmp(op,"!=")==0))==0)
printf("Expression is error");
else
{
printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);
addr++;
printf("\n%d\t T:=0",addr);
addr++;
printf("\n%d\t goto %d",addr,addr+2);
addr++;
printf("\n%d\t T:=1",addr);
}
break;
case 4:
exit(0);
}
}
}
void pm()
{
strrev(exp);
j=l-i-1;
```

```
strncat(exp1,exp,j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
}
void div()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
void plus()
{
strncat(exp1,exp,i+2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

OUTPUT



```
3.relational
4.Exit
Enter the choice:1

Enter the expression with assignment operator:a=b
Three address code:
temp=b
a=temp

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:2

Enter the expression with arithmetic operator:a=b+c
Three address code:
temp=a=b+c
temp1=temp

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:
```

5. Design LALR bottom up parser for the given language

filename calclex.l

```
%option noyywrap
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ { yylval=atoi(yytext); return NUMBER; }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
```

filename calc1.y

```
%{
 #include<stdio.h>
 int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{ printf("\nResult=%d\n",$$);return 0; };
E:E'+'E {$$=$1+$3;}
|E'-'E {$$=$1-$3;}
|E'*'E {$$=$1*$3;}
|E'/'E {$$=$1/$3;}
|E'%'E {$$=$1%$3;}
|'('E')' {$$=$2;}
| NUMBER {$$=$1;}
;
%%
void main()
{
 printf("\nEnter Any Arithmetic Expression:\n");
 yyparse();
 if(flag==0)
 printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
```

```
{
 printf("\nEntered arithmetic expression is Invalid\n\n");
 flag=1;
}
```

Output

```
$lex calclex.l
$yacc –d calc1.y
$cc lex.yy.c y.tab.c
$a.exe
Enter Any Arithmetic Expression
2+3
result=5
Entered arithmetic expression is Valid
```