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

```
/* program name is lexp.l */
%{
/* program to recognize a c program */
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
/*{printf("\n\n\t%s is a COMMENT\n",yytext);}*/
```

```
"*/" {COMMENT = 0;}
/* printf("\n\n\t%s is a COMMENT\n",yytext);}*/
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n BLOCK ENDS");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}
\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}
%%
int main(int argc,char **argv)
{
if (argc > 1)
{
FILE *file;
file = fopen(argv[1],"r");
if(!file)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
yyin = file;
}
yylex();
printf("\n\n");
```

```
return 0;
} int yywrap()
{
return 0;
}
```

Input:

```
$vi var.c
#include<stdio.h>
main()
{
int a,b;
}
$lex lex.l
$cc lex.yy.c
$./a.out var.c
```

OUTPUT:

```
#include<stdio.h> is a PREPROCESSOR DIRECTIVE
FUNCTION
main ()
BLOCK BEGINS
int is a KEYWORD
a IDENTIFIER
b IDENTIFIER
BLOCK ENDS
```

2. To Write a Program to compute the FIRST of a given grammar.

Program:

```
#include<stdio.h>
#include<ctype.h>
int main()
{
    int i,j,k,n;
    char str[10][10],f;
    printf("Enter no of productions: ");
    scanf("%d",&n);
    printf("Enter grammar\n");
    for(i=0;i<n;i++)
    {
        scanf("%s",&str[i]);
    }
    for(i=0;i<n;i++)
    {
        f= str[i][0];
        int temp=i;
        if(isupper(str[i][3]))
        {
            repeat:
                for(k=0;k<n;k++)
                {
                    if(str[k][0]==str[i][3])
                    {
                        if(isupper(str[k][3]))
                        {
                            i=k;
                            goto repeat;
                        }
                        else
                        {
                            printf("First(%c)=%c\n",f,str[k][3]);
                        }
                    }
                }
        }
        else
        {
            printf("First(%c)=%c\n",f,str[i][3]);
        }
        i=temp;
    }
}
```

Output:

Enter the number of productions
3
Enter grammar
S->AB
A->a
B->b
First(S)=a First(A)=a First(B)=b

3. To Write a Program to compute the FOLLOW of a given grammar

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{
    int np,i,j,k;
    char prod[30][30],fllow[10][10],lmad[10][10];
    printf("Enter no of productions: ");
    scanf("%d",&np);
    printf("Enter grammer: ");
    for (int i = 0; i < np; i++)
    {
        scanf("%s",&prod[i]);
    }
    for(i=0;i<np;i++)
    {
        if(i==0)
        {
            printf("Follow(%c)=$\n",prod[0][0]);
        }
        for(j=3;prod[i][j]!='\0';j++)
        {
            int temp2 = j;
            if(prod[i][j] >= 'A' && prod[i][j] <= 'Z')
            {
                if((strlen(prod[i])-1)==j)
                {
                    printf("Follow(%c)=Follow(%c)\n",prod[i][j],prod[i][0]);
                }
                int temp = i;
                char f = prod[i][j];
                if(!isupper(prod[i][j+1]) && (prod[i][j+1]!='\0')){
                    printf("Follow(%c) = %c\n",f,prod[i][j+1]);
```

```
                }
                if(isupper(prod[i][j+1]))
                {
                    repeat:
                    for(k=0;k<np;k++)
                    {
                        if(prod[k][0]==prod[i][j+1])
                        {
                            if(!isupper(prod[k][3]))
                            {
                                printf("Follow(%c)=%c\n",f,prod[k][3]);
                            }
                            else{
                                i=k;
                                j=2;
                                goto repeat;
                            }
                        }
                    }
                }
                i=temp;
            }
            j=temp2;
        }
    }
}
```

Output:

enter no. of productions
3
enter grammar
S->AB
A->a
B->b
Follow(S) = $
Follow(A)=b
Follow(B)=Follow(S)

4. To Write a Program to construct PREDICTIVE LL(1) TABLE

Program:

```
#include<stdio.h>
#include<string.h>
#include<process.h>

char prod[20][20],start[2];
char nonterm[20],term[20];
char input[10],stack[50];
int table[10][10];
int te,nte,np;

void init();
void parse();
void main()
{
    init();
    parse();
}

void init()
{
    int i,j;
    printf("Enter no of Terminals: ");
    scanf("%d",&te);
    for(i=0;i<te;i++)
    {
        printf("Enter terminal %d: ",i+1);
        scanf(" %c",&term[i]);
    }
    term[i]='$';
    printf("Enter no of non terminals: ");
    scanf("%d",&nte);
    for(i=0;i<nte;i++)
    {
        printf("Enter non-terminal %d: ",i+1);
        scanf(" %c",&nonterm[i]);
    }
    printf("Enter no of productions: ");
    scanf("%d",&np);
    printf("Enter Grammer: \n");
    for(i=0;i<np;i++)
    {
        scanf("%s",prod[i]);
```

```
    }
    start[0] = prod[0][0];
    printf("Enter input string: ");
    scanf("%s",input);
    printf("The grammer is : \n");
    for(int i=0;i<np;i++)
    {
        printf("%d\t%s\n",i+1,prod[i]);
    }
    printf("Enter Parsing table: \n");
    for(int i=0;i<nte;i++)
    {
        for(int j=0;j<=te;j++)
        {
            printf("Enter the Entry for table[%c][%c]: ",nonterm[i],term[j]);
            scanf("%d",&table[i][j]);
        }
    }
}
void parse()
{
    int i,j,prodno;
    int top=-1,cur=0;
    stack[++top] = '$';
    stack[++top] = start[0];
    do{
        if((stack[top]==input[cur]) && (input[cur]=='$'))
        {
            printf("\nThe given input string is parsed");
            exit(0);
        }
        else if(stack[top]==input[cur])
        {
            top--;
            cur++;
        }
        else if(stack[top]>='A' && stack[top]<= 'Z')
        {
            for(i=0;i<nte;i++)
            {
                if(nonterm[i]==stack[top])
                    break;
            }
            for(j=0;j<=te;j++)
            {
```

```
                if(term[j]==input[cur])
                    break;
            }
            prodno = table[i][j];
            if(prodno == 0)
            {
                printf("\nThe input String is not parsed..");
                exit(0);
            }
            else
            {
                for(i=strlen(prod[prodno-1])-1;i>=3;i--)
                {
                    if(prod[prodno-1][i]!='@')
                        stack[top++] = prod[prodno-1][i];
                }
                top--;
            }
        }
        else{
            printf("\nThe input String is not parsed..");
            exit(0);
        }
    }while(1);
}
```

Output:

Enter no of Terminals: 2
Enter terminal 1: a
Enter terminal 2: b

Enter no of non-terminals: 3
Enter non-terminal 1: S
Enter non-terminal 2: A
Enter non-terminal 3: B

Enter no of productions: 7
Enter Grammer:
S->aAB
S->bA
S->@
A->aAB
A->@
B->bB
B->@

Enter input string: aab$

The grammer is:
1 S->aAB
2 S->bA
3 S->@
4 A->aAB
5 A->@
6 B->bB
7 B->@

Enter Parsing table:
Enter the Entry for table[S][a]: 1
Enter the Entry for table[S][b]: 2
Enter the Entry for table[S][$]: 3
Enter the Entry for table[A][a]: 4
Enter the Entry for table[A][b]: 5
Enter the Entry for table[A][$]: 5
Enter the Entry for table[B][a]: 0
Enter the Entry for table[B][b]: 6
Enter the Entry for table[B][$]: 7

The given input string is parsed

5. To Implement Predictive Parsing algorithm

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
char nt[]={'E','A','T','B','F'},ter[]={'i','+','*','(',')','$'};
char arr[20][20][20]={
{"TA"," "," ","TA"," "," "},
{" ","+TA"," "," ","#","#"},
{"FB"," "," ","FB"," "," "},
{" ","#","*FB"," ","#","#"},
{"i"," "," ","(E)"," "," "}
};
char ipstr[20];
char stack[40],prod[10];
int i=0,top=1,ia,ix;
void pop();
void push(char );
int resolve_nt(char );
```

```
int resolve_t(char );
void advance();
void main(void )
{
    char a,x;
    int len,temp,k;
    stack[0]='$';
    stack[1]='E';
    printf("Enter the input string(Enter $ as an end marker): ");
    scanf("%s",ipstr);
    printf("I/P String\tStack Contents\t\tProduction Used\n");
    while(1)
    {
        a=ipstr[i];
        x=stack[top];
        for(k=i;ipstr[k]!='$';k++)
            printf("%c",ipstr[k]);

        printf("$\t\t");
        if(x==a)
        {
            if(x=='$')
            {
                printf("\rinput string is accepted");
                break;
            }
            else
            {
                pop();
                advance();
            }
        }
        else if(isupper(x))
        {
            ix=resolve_nt(x);
            ia=resolve_t(a);
            strcpy(prod,arr[ix][ia]);
            len=strlen(prod);
            pop();
            for(k=1;k<=len;k++)
                push(prod[len-k]);
            if(stack[top]=='#')
                pop();
        }
        else
```

```
        {
            printf("Error: Could not parse teh input string");
            break;
        }
        /*To display the stack contents and the production used*/
        for(k=0;k<=top;k++)
            printf("%c",stack[k]);
        printf("\t\t\t\t%s\n",prod);
    }

}

void push(char t)
{
    stack[++top]=t;
}

void pop()
{
    top--;
}

void advance()
{
    i++;
}

int resolve_nt(char t)
{
    int k,index;
    for(k=0;k<5;k++)
    {
        if(t==nt[k])
            return k;
    }
}

int resolve_t(char t)
{
    int k,index;
    for(k=0;k<6;k++)
    {
        if(t==ter[k])
            return k;
    }
```

```
}
```

Output:

Enter the input string (Enter $ as an end marker): i+i$

I/P String	Stack Contents	Production Used
i+i$	$AT	TA
i+i$	$ABF	FB
i+i$	$ABi	i
i+i$	$AB	i
+i$	$A	#
+i$	$AT+	+TA
+i$	$AT	+TA
i$	$ABF	FB
i$	$ABi	i
i$	$AB	i
$	$A	#
$	$	#

input string is accepted

6. To write a C program to generate three address code.

Program:

```
#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];
int main()
{
    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)||(strcmp(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:

1.assignment

2.arithmetic
3.relational
4.Exit
Enter the choice:1
Enter the expression with assignment operator: a=10
Three address code:
temp=10
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
temp1=temp+c

1.assignment
2.arithmetic
3.relational
4.Exit
Enter the choice:3
Enter the expression with relational operator: a > b
100 if a>b goto 103
101 T:=0
102 goto 104
103 T:=1

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

7. To Implement SLR(1) Parsing algorithm

Program:

```
#include<stdio.h>
#include<string.h>
int axn[][6][2]={
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
    {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
```

```
    {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{-1,-1},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
    {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,1},{-1,-1}},
    {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
    {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
    {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};//Axn Table
int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1, -1,9,3,-1,-
1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}; //GoTo table
int a[10];
char b[10];
int top=-1,btop=-1,i;
void push(int k)
{
    if(top<9)
        a[++top]=k;
}
void pushb(char k)
{
    if(btop<9)
        b[++btop]=k;
}
char TOS()
{
    return a[top];
}
void pop()
{
    if(top>=0)
        top--;
}
void popb()
{
    if(btop>=0)
        b[btop--]='\0';
}
void display()
{
    for(i=0;i<=top;i++)
        printf("%d%c",a[i],b[i]);
}
void display1(char p[],int m) //Displays The Present Input String
```

```
{
    int l;
    printf("\t\t");
    for(l=m;p[l]!='\0';l++)
    printf("%c",p[l]);
    printf("\n");
}
void error()
{
    printf("Syntax Error");
}
void reduce(int p)
{
    int len,k,ad;
    char src,*dest;
    switch(p)
    {
        case 1:dest="E+T";
            src='E';
            break;
        case 2:dest="T";
            src='E';
            break;
        case 3:dest="T*F";
            src='T';
            break;
        case 4:dest="F";
            src='T';
            break;
        case 5:dest="(E)";
            src='F';
            break;
        case 6:dest="i";
            src='F';
            break;
        default:dest="\0";
            src='\0';
            break;
    }
    for(k=0;k<strlen(dest);k++)
    {
        pop();
        popb();
    }
    pushb(src);
```

```
    switch(src)
    {
        case 'E':ad=0;
        break;
        case 'T':ad=1;
        break;
        case 'F':ad=2;
        break;
        default: ad=-1;
        break;
    }
    push(gotot[TOS()][ad]);
}
int main()
{
    int j,st,ic;
    char ip[20]="\0",an;
    // clrscr();
    printf("Enter any String\n");
    scanf("%s",ip);
    push(0);
    display();
    printf("\t%s\n",ip);
    for(j=0;ip[j]!='\0';)
    {
        st=TOS();
        an=ip[j];
        if(an>='a'&&an<='z') ic=0;
        else if(an=='+') ic=1;
        else if(an=='*') ic=2;
        else if(an=='(') ic=3;
        else if(an==')') ic=4;
        else if(an=='$') ic=5;
        else {
            error();
            break;
        }
        if(axn[st][ic][0]==100)
        {
            pushb(an);
            push(axn[st][ic][1]);
            display();
            j++;
            display1(ip,j);
        }
```

```
        if(axn[st][ic][0]==101)
        {
            reduce(axn[st][ic][1]);
            display();
            display1(ip,j);
        }
        if(axn[st][ic][1]==102)
        {
            printf("Given String is accepted \n");
            break;
        }
    }
    return 0;
}
```

Output:

Enter any String
a+a*a$
0 a+a*a$
0a5 +a*a$
0F3 +a*a$
0T2 +a*a$
0E1 +a*a$
0E1+6 a*a$
0E1+6a5 *a$
0E1+6F3 *a$
0E1+6T9 *a$
0E1+6T9*7 a$
0E1+6T9*7a5 $
0E1+6T9*7F10 $
0E1+6T9 $
0E1 $
Given String is accepted

8. To Design LALR bottom-up parser for the given language

Program:

<parser.l>

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
```

```
[0-9]+ {yylval.dval=atof(yytext);
return DIGIT;
}
\n|. return yytext[0];
%%
<parser.y>
%{
/*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
#include<stdio.h>
%}
%union
{
double dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line: expr '\n' {
printf("%g\n",$1);
}

;
expr: expr '+' term {$$=$1 + $3 ;}
| term
;
term: term '*' factor {$$=$1 * $3 ;}
| factor
;
factor: '(' expr ')' {$$=$2 ;}
| DIGIT
;
%%
int main()
{
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

INPUT:

```
$lex parser.l
$yacc –d parser.y
$cc lex.yy.c y.tab.c –ll –lm
$./a.out
```

OUTPUT:
2+3
5.0000