1. Design a lexical analyzer for a given language and the lexical analyzer should ignore redundant spaces tabs and new lines.It should also ignore comments..Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some resonable value. Simulate the same in C language.

Ans:-

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp("int",str)==0||strcmp
("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("static",str)==0||strcmp("swit
ch",str
)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}
int main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("\nEnter the c program");
gets(st1);
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","w");
```

```
while((c=getc(f1))!=EOF) {
if(isdigit(c))
{
tokenvalue=c-'0';
c=getc(f1);
while(isdigit(c)) {
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else
if(isalpha(c))
{
putc(c,f2);
c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else
if(c==' '||c=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j<i;j++)
```

```
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF) {
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0; }
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
}
```

Input:-

```
{
int a[3],t1,t2;
t1=2; a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then
print(t2);
else {
int t3;
t3=99;
t2=-25;
print(-t1+t2*t3); /* this is a comment on 2 lines */
} endif
}
```

Output:-

```
Variables : a[3] t1 t2 t3
Operator : - + * / >
Constants : 2 1 3 6 5 99 -25
Keywords : int if then else endif
```

```
Special Symbols : , ; ( ) { }
Comments: this is a comment on 2 lines
```

2. Write a C program to identify whether a given line is a comment or not.

Ans:-

```
#include <stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main() {
char com[30];
int i=2,a=0;

printf("\n Enter comment:");
gets(com);
if(com[0]=='/') {
if(com[1]=='/')
printf("\n It is a comment");
else if(com[1]=='*') {
for(i=2;i<=30;i++)
{
if(com[i]=='*'&&com[i+1]=='/')
{
printf("\n It is a comment");
a=1;
break; }
else
continue; }
if(a==0)
printf("\n It is not a comment");
}
else
printf("\n It is not a comment");
}
else
printf("\n It is not a comment");
getch();
```

```
}

Output:-
Enter comment://hi

It is a comment

Enter comment:hi

It is not a comment
```

3. Write a C program to recognize strings under 'a' 'a*b+' 'abb'.

Ans:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
char s[20],c;
int state=0,i=0;

printf("\n Enter a string:");
gets(s);
while(s[i]!='\0')
{
switch(state)
{
case 0: c=s[i++];
if(c=='a') state=1;
else if(c=='b')
state=2;
else state=6;
```

```
break;
case 1: c=s[i++];
if(c=='a')
state=3;
else if(c=='b')
state=4;
else
state=6;
                break;
case 2: c=s[i++];
if(c=='a') state=6;
else if(c=='b')
state=2;
else state=6;
break;
case 3: c=s[i++]; if(c=='a')
state=3;
else if(c=='b')
state=2;
else
state=6;
break;
case 4: c=s[i++];
                if(c=='a')
state=6;
else if(c=='b')
state=5;
else state=6;
break;
case 5: c=s[i++];
if(c=='a')
state=6;
else if(c=='b')
```

```
state=2;
else
state=6;
break;
case 6: printf("\n %s is not recognised.",s);
exit(0);
} }
if(state==1)
printf("\n %s is accepted under rule 'a'",s);
else if((state==2)||(state==4))
printf("\n %s is accepted under rule 'a*b+'",s);
else if(state==5)
printf("\n %s is accepted under rule 'abb'",s); getch();
}
```

Output:-

```
Enter a string:aaabbb
aaabbb is accepted under rule 'a*b+'

Enter a string:aabcc
aabcc is not recognised.
```

4. Write a C program to test whether a given identifier is valid or not.

Ans:-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
```

```
   char a[10];
   int flag,
   i=1;

   printf("\n Enter an identifier:");
   gets(a);
   if(isalpha(a[0]))
   flag=1;
   else
   printf("\n Not a valid identifier");
   while(a[i]!='\0')
   {
      if(!isdigit(a[i])&&!isalpha(a[i]))
      {
         flag=0; break;

      } i++;

   }
   if(flag==1)
   printf("\n Valid identifier");
   getch();

}
```

```
Output:-
Enter an identifier:last
Valid identifier

Enter an identifier:1yf
Not a valid identifier
```

5. Write a C program to simulate lexical analyser for validating operators.

Ans:-

```
#include <stdio.h>
#include<string.h>
#include<conio.h>
#include<ctype.h>
void main()
{
char s[5];

printf("\n Enter any operator:");
gets(s);
switch(s[0])
{
case'>': if(s[1]=='=')
printf("\n Greater than or equal");
else
printf("\n Greater than");
break;
case'<': if(s[1]=='=')
printf("\n Less than or equal");
else
printf("\nLess than");
break;
case'=': if(s[1]=='=')
printf("\nEqual to");
else
printf("\nAssignment");
break;
case'!': if(s[1]=='=')
printf("\nNot Equal");
else
printf("\n Bit Not");
```

```
break;
case'&': if(s[1]=='&')
printf("\nLogical AND");
else
printf("\n Bitwise AND");
break;
case'|': if(s[1]=='|')
printf("\nLogical OR");
else
printf("\nBitwise OR");
break;
case'+': printf("\n Addition");
break;
case'-': printf("\nSubstraction");
break;
case'*': printf("\nMultiplication");
break;
case'/': printf("\nDivision");
break;
case'%': printf("Modulus");
break;
default: printf("\n Not a operator");
}
getch();
}
```

Output:-

```
Enter any operator:+
Addition
Enter any operator:%
Modulus
```

6. Implement lexical analyser using Jlex, flex and other lexical analyser generating tools.

Ans:-

```
/* 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

main()

{

```
int a,b;

}
```

Output:-

```
$lex lex.l

$cc lex.yy.c

$./a.out var.c

#include is a PREPROCESSOR DIRECTIVE

FUNCTION

main (

)

BLOCK BEGINS

int is a KEYWORD

a IDENTIFIER

b IDENTIFIER

BLOCK ENDS
```

7. write a C program for implementing the functionalities of predictive parser.

Ans:-

```
#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];
int 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:-

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->@

8. write a C program for constructing of LL (1) parsing.

Ans:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s[20],stack[20];
void main()
{
```

```
    char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," "," ","n","*fc"," a ","n","n","i"," "," ","(e)"," "," "};
    int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
    int i,j,k,n,str1,str2;

    printf("\n Enter the input string: ");
    scanf("%s",s);
    strcat(s,"$");
    n=strlen(s);
    stack[0]='$';
    stack[1]='e';
    i=1;
    j=0;
    printf("\nStack Input\n");
    printf("______\n");
    while((stack[i]!='$')&&(s[j]!='$'))
    {
       if(stack[i]==s[j])
       {
          i--;
          j++;
       }
       switch(stack[i])
       {
          case 'e':
          str1=0;
          break;
          case 'b':
          str1=1;
          break;
          case 't':
          str1=2;
          break;
```

```
    case 'c':
    str1=3;
    break;
    case 'f':
    str1=4;
    break;

}
switch(s[j])
{ case 'i':
str2=0;
break;
case '+':
str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;

}
if(m[str1][str2][0]=='\0')
{
    printf("\nERROR");
    exit(0);

}
else if(m[str1][str2][0]=='n')
i--;
```

```
        else if(m[str1][str2][0]=='i')
        stack[i]='i';
        else
        {
           for(k=size[str1][str2]-1;k>=0;k--)
           {
              stack[i]=m[str1][str2][k];
              i++;

           }
           i--;

        }
        for(k=0;k<=i;k++)
        printf(" %c",stack[k]);
        printf(" ");
        for(k=j;k<=n;k++)
        printf("%c",s[k]);
        printf(" \n ");

    }
    printf("\n SUCCESS");
    getch();

}
```

Output:-

Enter the input string: i*i+i

Stack Input

———

$ b t i*i+i$

$ b c f i*i+i$

$ b c i i*i+i$

$ b c f * *i+i$

$ b c i i+i$

$ b +i$

$ b t + +i$

$ b c f i$

$ b c i i$

$ b $

SUCCESS

9. write a C program to implement LALR parsing.

Ans:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void push(char *,int *,char);

char stacktop(char *);

void isproduct(char,char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char,char);
```

```
char pop(char *,int *);

void printt(char *,int *,char [],int);

void rep(char [],int);

struct action

{

char row[6][5];

};

const struct action A[12]={

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","emp","acc"},

{"emp","rc","sh","emp","rc","rc"},

{"emp","re","re","emp","re","re"},

{"sf","emp","emp","se","emp","emp"},

{"emp","rg","rg","emp","rg","rg"},

{"sf","emp","emp","se","emp","emp"},

{"sf","emp","emp","se","emp","emp"},
```

```
{"emp","sg","emp","emp","sl","emp"},

{"emp","rb","sh","emp","rb","rb"},

{"emp","rb","rd","emp","rd","rd"},

{"emp","rf","rf","emp","rf","rf"}

};

struct gotol

{

char r[3][4];

};

const struct gotol G[12]={

{"b","c","d"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"i","c","d"},

{"emp","emp","emp"},

{"emp","j","d"},
```

```
{"emp","emp","k"},

{"emp","emp","emp"},

{"emp","emp","emp"},

};

char ter[6]={'i','+','*',')','(','$'};

char nter[3]={'E','T','F'};

char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};

char stack[100];

int top=-1;

char temp[10];

struct grammar

{
char left;

char right[5];

};

const struct grammar rl[6]={

{'E',"e+T"},
```

```
{'E',"T"},

{'T',"T*F"},

{'T',"F"},

{'F',"(E)"},

{'F',"i"},

};

void main()

{

char inp[80],x,p,dl[80],y,bl='a';

int i=0,j,k,l,n,m,c,len;



printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);

inp[len]='$';

inp[len+1]='\0';
```

```
push(stack,&top,bl);

printf("\n stack \t\t\t input");

printt(stack,&top,inp,i);

do

{

x=inp[i];

p=stacktop(stack);


isproduct(x,p);

if(strcmp(temp,"emp")==0)

error();

if(strcmp(temp,"acc")==0)

break;

else

{

if(temp[0]=='s')

{
```

```
push(stack,&top,inp[i]);

push(stack,&top,temp[1]);

i++;
}

else

{

if(temp[0]=='r')

{

j=isstate(temp[1]);

strcpy(temp,rl[j-2].right);

dl[0]=rl[j-2].left;

dl[1]='\0';

n=strlen(temp);

for(k=0;k<2*n;k++)

pop(stack,&top);

for(m=0;dl[m]!='\0';m++)

push(stack,&top,dl[m]);
```

```
l=top;

y=stack[l-1];

isreduce(y,dl[0]);

for(m=0;temp[m]!='\0';m++)

push(stack,&top,temp[m]);

}

}

}

printf(stack,&top,inp,i);

}while(inp[i]!='\0');

if(strcmp(temp,"acc")==0)

printf(" \n accept the input ");

else

printf(" \n do not accept the input ");

getch();

}
void push(char *s,int *sp,char item)
```

```
{

if(*sp==100)

printf(" stack is full ");

else

{

*sp=*sp+1;
s[*sp]=item;

}

}

char stacktop(char *s)

{

char i;

i=s[top];

return i;

}

void isproduct(char x,char p)

{
```

```
int k,l;

k=ister(x);

l=isstate(p);

strcpy(temp,A[l-1].row[k-1]);

}

int ister(char x)

{

int i;

for(i=0;i<6;i++)

if(x==ter[i])

return i+1;

return 0;

}
int isnter(char x)

{

int i;

for(i=0;i<3;i++)
```

```
if(x==nter[i])

return i+1;

return 0;

}

int isstate(char p)

{

int i;

for(i=0;i<12;i++)

if(p==states[i])
return i+1;

return 0;

}

void error()

{

printf(" error in the input ");

exit(0);

}
```

```
void isreduce(char x,char p)

{

int k,l;

k=isstate(x);

l=isnter(p);

strcpy(temp,G[k-1].r[l-1]);

}
char pop(char *s,int *sp)

{

char item;

if(*sp==-1)

printf(" stack is empty ");

else

{

item=s[*sp];

*sp=*sp-1;

}
```

```
return item;

}

void printt(char *t,int *p,char inp[],int i)

{

int r;

printf("\n");

for(r=0;r<=*p;r++)

rep(t,r);

printf("\t\t\t");

for(r=i;inp[r]!='\0';r++)
printf("%c",inp[r]);

}
void rep(char t[],int r)

{

char c;

c=t[r];

switch(c)

{
```

```
case 'a': printf("0");

break;

case 'b': printf("1");

break;

case 'c': printf("2");

break;

case 'd': printf("3");

break;

case 'e': printf("4");

break;

case 'f': printf("5");

break;

case 'g': printf("6");

break;

case 'h': printf("7");

break;
```

```
case 'm': printf("8");

break;

case 'j': printf("9");

break;

case 'k': printf("10");

break;

case 'l': printf("11");

break;

default :printf("%c",t[r]);

break;

}

}
```

Output

Stack input

```
0    i*i+i$
0i5  *i+i$
0F3   *i+i$
0T2   *i+i$
0T2   *7 i+i$
0T2   *7i5 +i$
0T2  *7i5F10 +i$
```

0T2 +i$

0E1 +i$

0E1 +6 i$

0E1 +6i5 $

0E1 +6F3 $

0E1 +6T9 $

0E1 $

accept the input*/

10. Write a C program to implement operator precedence parsing.

Ans:-

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


char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={


        /*stack   +   -   *   /   ^   i   (   )   $  */

        /*  + */  '>', '>','<','<','<','<','<','>','>',

        /*  - */  '>', '>','<','<','<','<','<','>','>',
```

```
        /*  * */  '>', '>','>','>','<','<','<','>','>',

        /*  / */  '>', '>','>','>','<','<','<','>','>',

        /*  ^ */  '>', '>','>','>','<','<','<','>','>',

        /*  i */  '>', '>','>','>','>','e','e','>','>',

        /*  ( */  '<', '<','<','<','<','<','<','>','e',

        /*  ) */  '>', '>','>','>','>','e','e','>','>',

        /*  $ */  '<', '<','<','<','<','<','<','<','>',

            };

int getindex(char c)
{
switch(c)
   {
   case '+':return 0;
   case '-':return 1;
   case '*':return 2;
   case '/':return 3;
   case '^':return 4;
   case 'i':return 5;
   case '(':return 6;
   case ')':return 7;
   case '$':return 8;
   }
}
```

```
int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}


int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
   {
   len=strlen(handles[i]);
   if(stack[top]==handles[i][0]&&top+1>=len)
      {
      found=1;
      for(t=0;t<len;t++)
         {
         if(stack[top-t]!=handles[i][t])
            {
            found=0;
            break;
            }
         }
      if(found==1)
         {
         stack[top-t+1]='E';
         top=top-t+1;
         strcpy(lasthandle,handles[i]);
         stack[top+1]='\0';
         return 1;//successful reduction
         }
      }
```

```
   }
return 0;
}



void dispstack()
{
int j;
for(j=0;j<=top;j++)
    printf("%c",stack[j]);
}



void dispinput()
{
int j;
for(j=i;j<l;j++)
    printf("%c",*(input+j));
}



void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
```

```
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
{
shift();
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tShift");
if(prec[getindex(stack[top])][getindex(input[i])]=='>')
{
while(reduce())
{
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tReduced: E->%s",lasthandle);
}
}
}

if(strcmp(stack,"$E$")==0)
   printf("\nAccepted;");
else
   printf("\nNot Accepted;");
}
```

INPUT & OUTPUT:

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

2+3
5.0000

Enter the string

$lex parser.l

STACK INPUT ACTION

$$ lex$ Shift