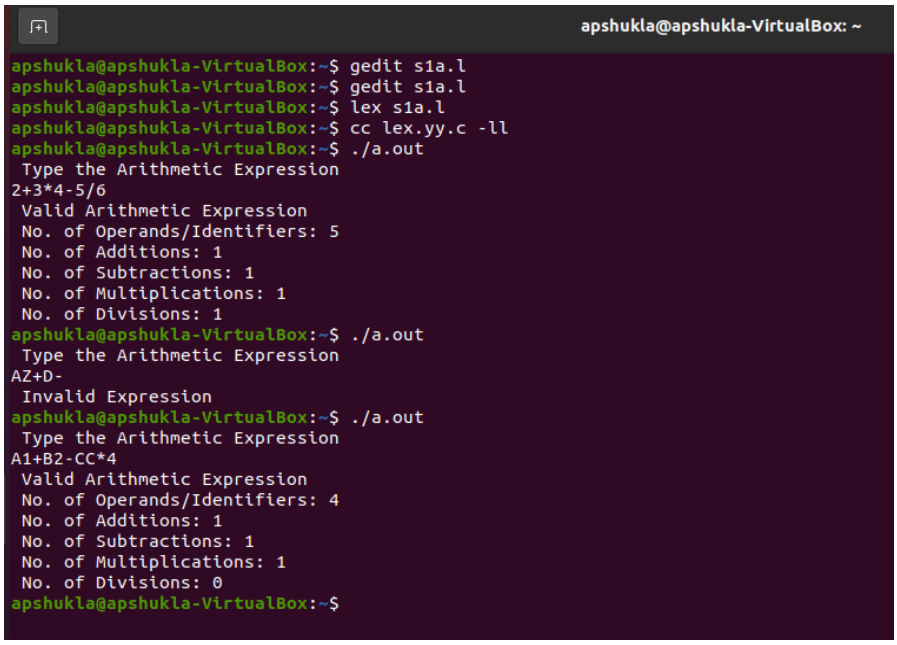
System Software Laboratory Manual

1.a. Write a LEX program to recognize valid *arithmetic expression.* Identifiers in the expression could be only integers and operators could be + and *. Count the identifiers & operators present and print them separately.

```
%{
int a[]={0,0,0,0}, i, valid=1, opnd=0;
int ext();
%}
%X OPER
%%
[a-zA-Z0-9]+  { BEGIN OPER; opnd++; }
<OPER>"+"   { if(valid)      { valid=0; i=0; }      else     ext(); }
<OPER>"-"   { if(valid)      { valid=0; i=1; }      else     ext(); }
<OPER>"*"   { if(valid)      { valid=0; i=2; }      else     ext(); }
<OPER>"/"   { if(valid)      { valid=0; i=3; }      else     ext(); }
<OPER>[a-zA-Z0-9]+          { opnd++;
                        if(valid==0)
                        {
                                valid=1; a[i]++;
                        }
                        else
                                ext();
                        }
<OPER>"\n"  { if(valid==0)
                                ext();
                  else
                                return 0;
                  }
.\n ext();
%%
int ext()
```

```
{
        printf(" Invalid Expression \n");
        exit(0);
}
int main()
{
printf(" Type the Arithmetic Expression \n");
yylex();
printf(" Valid Arithmetic Expression \n");
printf(" No. of Operands/Identifiers: %d \n", opnd);
printf(" No. of Additions: %d \n No. of Subtractions: %d \n", a[0], a[1]);
printf(" No. of Multiplications: %d \n No. of Divisions: %d \n", a[2], a[3]);
}
```

Output: -



1. b. Write YACC program to evaluate *arithmetic expression* involving operators: +, -, *, and /

Lex Code: -

```
%{
#include "y.tab.h"
extern int yylval;
```

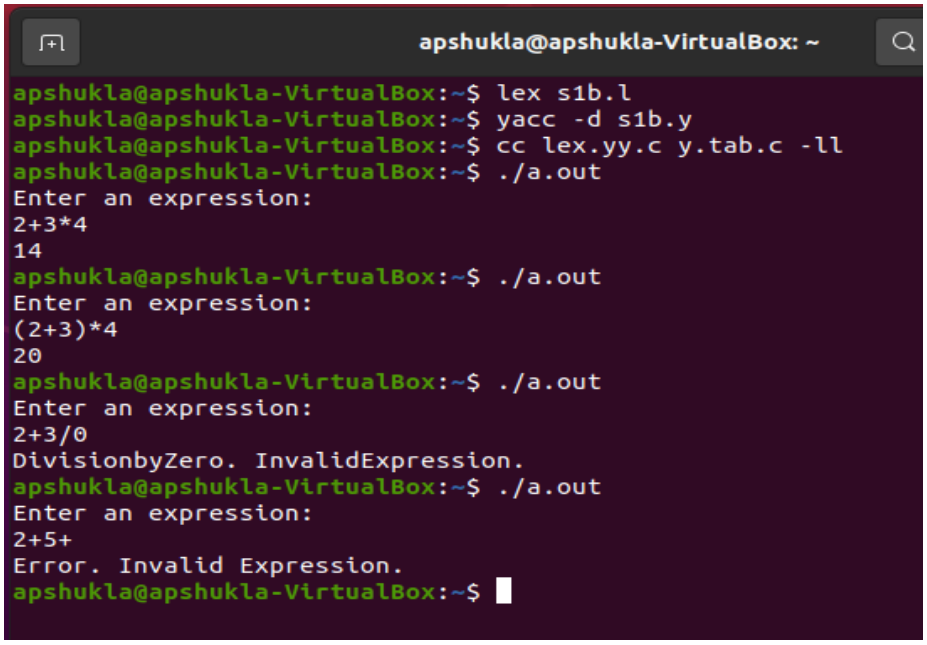
```
%}

%%
[0-9]+          { yylval=atoi(yytext); return num; }
[\+\-\*\/]      { return yytext[0]; }
[)]             { return yytext[0]; }
[(]             { return yytext[0]; }
.               { ; }
\n              { return 0; }
%%
```

Yacc Code: -

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
int yyerror();
%}

%token num
%left '+' '-'
%left '*' '/'

%%
input: exp      { printf("%d\n",$$); exit(0); }
exp: exp'+'exp { $$=$1+$3; }
|exp'-'exp      { $$=$1-$3; }
|exp'*'exp      { $$=$1*$3; }
|exp'/'exp      { if($3==0) { printf("DivisionbyZero. InvalidExpression.\n"); exit(0); } else {
$$=$1/$3; } }
|'('exp')'      { $$=$2; }
|num            { $$=$1; };
```

```
%%


int yyerror()
{
        printf("Error. Invalid Expression.\n");
        exit(0);
}
int main()
{
        printf("Enter an expression: \n");
        yyparse();
}
```

Output: -



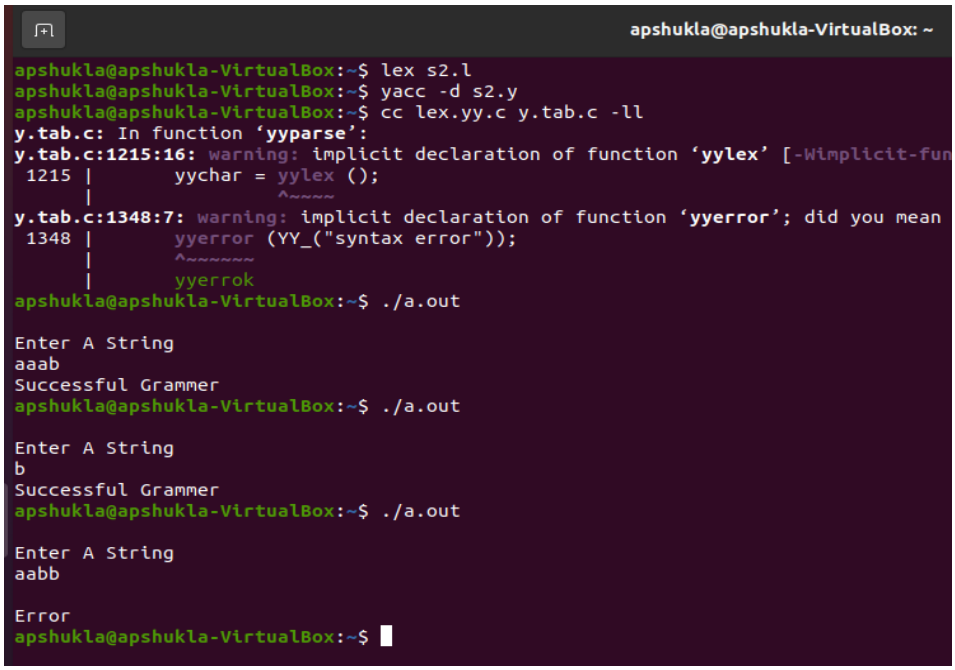
2. Develop, Implement and Execute a program using YACC tool to recognize all strings ending with *b* preceded by *n a's* using the grammar *an b* (note: input *n* value)

Lex Code: -

```
%{
#include "y.tab.h"
%}
%%
```

```
a       { return A; }
b       { return B; }
[\n]    { return '\n'; }
%%
Yacc code->
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B
%%
input: s'\n'       { printf("Successful Grammer\n"); exit(0); }
s: A s1 B | B
s1: ; | A s1
%%
int main()
{
        printf("\nEnter A String\n");
        yyparse();
}
int yyerror()
{
        printf("\nError\n");
        exit(0);
}
```

Output: -



3. Design, develop and implement YACC/C program to construct *Predictive / LL(1) Parsing Table* for the grammar rules: *A->aBa* , *B->bB* | e. Use this table to parse the sentence: *abba$*

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>


char prod [3][10]={"A->aBa","B->bB","B->@"};

char first[3][10]={"a","b","@"};

char follow[3][10]={"$","a","a"};

char table[3][4][10];


char input[10];

int top=-1;

char stack[25];

char curp[20];
```

```
void push(char item)
{
  stack[++top]=item;
}
void pop()
{
  top=top-1;
}
void display()
{
  int i;
  for(i=top;i>=0;i--)
  printf("%c",stack[i]);
}

int numr(char c)
{
  switch(c)
  {
    case'A':return 1;
    case'B':return 2;
    case'a':return 1;
    case'b':return 2;
    case'@':return 3;
  }
  return 1;
}

int main()
{
```

```
 char c;
 int i,j,k,n;
 for(i=0;i<3;i++){
  for(j=0;j<4;j++){
    strcpy(table[i][j],"EMPTY");
  }
 }
 printf("\nGrammar\n");

 for(i=0;i<3;i++)
 printf("%s\n",prod[i]);

 printf("\nfirst={%s,%s,%s}",first[0],first[1],first[2]);
 printf("\nfollow={%s,%s}\n",follow[0],follow[1]);
 printf("\nPredictive parsing table for the given grammar :\n");

 strcpy(table[0][0],"");
 strcpy(table[0][1],"a");
 strcpy(table[0][2],"b");
 strcpy(table[0][3],"$");
 strcpy(table[1][0],"A");
 strcpy(table[2][0],"B");

 for(i=0;i<3;i++)
 {
  if(first[i][0]!='@')
  strcpy(table[numr(prod[i][0])][numr(first[i][0])],prod[i]);
  else
  strcpy(table[numr(prod[i][0])][numr(follow[i][0])],prod[i]);
 }
```

```
 printf("\n----------------------------------------------------------------\n");
 for(i=0;i<3;i++){
  for(j=0;j<4;j++)
  {
    printf("%-30s",table[i][j]);
    if(j==3) printf("\n----------------------------------------------------------------\n");
  }
 }

 printf("Enter the input string terminated with $ to parse:-");
 scanf("%s",input);
 for(i=0;input[i]!='\0';i++){
  if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='$'))
  {
    printf("Invalid String");
    exit(0);
  }
 }

        if(input[i-1]!='$')
 {
  printf("\n\nInput String Entered Without End Marker $\n");
  exit(0);
 }

        push('$');
 push('A');
 i=0;

        printf("\n\n");
```

```
 printf("Stack\t Input\tAction");
 printf("\n-----------------------------------------------------------------\n");

        while(input[i]!='$'&&stack[top]!='$')
{
  display();
  printf("\t\t%s\t",(input+i));
  if(stack[top]==input[i])
  {
   printf("\tMatched %c\n", input[i]);
   pop();
   i++;
  }
  else
  {
   if(stack[top]>=65&&stack[top]<92)
   {
    strcpy(curp,table[numr(stack[top])][numr(input[i])]);
    if(!(strcmp(curp,"e")))
    {
     printf("\nInvalid String - Rejected\n");
     exit(0);
    }
    else
    {
     printf("\tApply production %s\n",curp);
     if(curp[3]=='@')
     pop();
     else
     {
```

```
            pop();
            n=strlen(curp);
            for(j=n-1;j>=3;j--)
            push(curp[j]);
          }
        }
      }
    }
  }

  display();
  printf("\t\t%s\t",(input+i));
  printf("\n---------------------------------------------------------------------\n");
  if(stack[top]=='$'&&input[i]=='$')
  {
    printf("\nValid String - Accepted\n");
  }
  else
  {
    printf("Invalid String - Rejected\n");
  }
}
```

Output: -

```
apshukla@apshukla-VirtualBox:~$ ./a.out

Grammar
A->aBa
B->bB
B->@

first={a,b,@}
follow={$,a}

Predictive parsing table for the given grammar :

------------------------------------------------------------------
                            a                           b                           $
------------------------------------------------------------------
A                           A->aBa                      EMPTY                       EMPTY
------------------------------------------------------------------
B                           B->@                        B->bB                       EMPTY
------------------------------------------------------------------
Enter the input string terminated with $ to parse:-abba$


Stack    Input  Action
------------------------------------------------------------------
A$              abba$           Apply production A->aBa
aBa$            abba$           Matched a
Ba$             bba$            Apply production B->bB
bBa$            bba$            Matched b
Ba$             ba$             Apply production B->bB
bBa$            ba$             Matched b
Ba$             a$              Apply production B->@
a$              a$              Matched a
$               $
------------------------------------------------------------------

Valid String - Accepted
```

```
apshukla@apshukla-VirtualBox:~$ ./a.out

Grammar
A->aBa
B->bB
B->@

first={a,b,@}
follow={$,a}

Predictive parsing table for the given grammar :

------------------------------------------------------------------
                            a                           b                           $
------------------------------------------------------------------
A                           A->aBa                      EMPTY                       EMPTY
------------------------------------------------------------------
B                           B->@                        B->bB                       EMPTY
------------------------------------------------------------------
Enter the input string terminated with $ to parse:-abba


Input String Entered Without End Marker $
```

4. Design, develop and implement YACC/C program to demonstrate *Shift Reduce Parsing*
technique for the grammar rules: *E ®E+T | T, T ®T*F | F, F ®(E) | id* and
parse the sentence: *id + id * id*.

#include<stdio.h>

```
#include<string.h>
int k=0, z=0, i=0, j=0, c=0;
char a[16], ac[20], stk[15], act[10];
void check();
int main()
{
   puts("Grammer is E->E+E \nE->E*E \nE->(E) \nE->id");
   puts("Enter input string");
   gets(a);
   c = strlen(a);
   strcpy(act, "SHIFT->");
   puts("stack\tinput\taction");
   for(k=0, i=0; j<c; k++, i++, j++)
   {
      if(a[j]=='i' && a[j+1]=='d')
      {
         stk[i]=a[j];
         stk[i+1]=a[j+1];
         stk[i+2]='\0';
         a[j]=' ';
         a[j+1]=' ';
         printf("\n$%s\t%s$\t%sid", stk, a, act);
         check();
      }
      else
      {
         stk[i]=a[j];
         stk[i+1]='\0';
         a[j]=' ';
         printf("\n$%s\t%s$\t%ssymbols", stk, a, act);
```

```
            check();
        }
    }
    if(stk[0]=='E' && stk[1]=='\0')
        printf("\nAccepted\n");
    else
        printf("\nError\n");
}

void check()
{
    strcpy(ac, "REDUCE TO E");
    for(z=0; z<c; z++)
        if(stk[z]=='i' && stk[z+1]=='d')
        {
            stk[z]='E';
            stk[z+1]='\0';
            printf("\n$%s\t%s$\t%s", stk, a, ac);
            j++;
        }

    for(z=0; z<c; z++)
        if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
        {
            stk[z]='E';
            stk[z+1]='\0';
            stk[z+2]='\0';
            printf("\n$%s\t%s$\t%s", stk, a, ac);
            i=i-2;
        }
```

```
    for(z=0; z<c; z++)
        if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
        {
            stk[z]='E';
            stk[z+1]='\0';
            stk[z+2]='\0';
            printf("\n$%s\t%s$\t%s", stk, a, ac);
            i=i-2;
        }

    for(z=0; z<c; z++)
        if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
        {
            stk[z]='E';
            stk[z+1]='\0';
            stk[z+2]='\0';
            printf("\n$%s\t%s$\t%s", stk, a, ac);
            i=i-2;
        }
}
```

Output: -

```
apshukla@apshukla-VirtualBox:~$ ./a.out
Grammer is E->E+E
E->E*E
E->(E)
E->id
Enter input string
id+id*id
stack   input   action

$id       +id*id$       SHIFT->id
$E        +id*id$       REDUCE TO E
$E+        id*id$       SHIFT->symbols
$E+id        *id$       SHIFT->id
$E+E         *id$       REDUCE TO E
$E           *id$       REDUCE TO E
$E*           id$       SHIFT->symbols
$E*id           $       SHIFT->id
$E*E            $       REDUCE TO E
$E              $       REDUCE TO E
Accepted
apshukla@apshukla-VirtualBox:~$ ./a.out
Grammer is E->E+E
E->E*E
E->(E)
E->id
Enter input string
id+id-id
stack   input   action

$id       +id-id$       SHIFT->id
$E        +id-id$       REDUCE TO E
$E+        id-id$       SHIFT->symbols
$E+id        -id$       SHIFT->id
$E+E         -id$       REDUCE TO E
$E           -id$       REDUCE TO E
$E-           id$       SHIFT->symbols
$E-id           $       SHIFT->id
$E-E            $       REDUCE TO E
Error
apshukla@apshukla-VirtualBox:~$
```

5. Design, develop and implement a C/Java program to generate the machine code using *Triples*
for the statement *A = -B * (C +D)* whose intermediate code in three-address form:
T1 = -B ***//Z=-B***
T2 = C + D ***//Y=C+D***
T3 = T1 + T2 ***//X=Z+Y***
A = T3 ***//A=X***

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
```

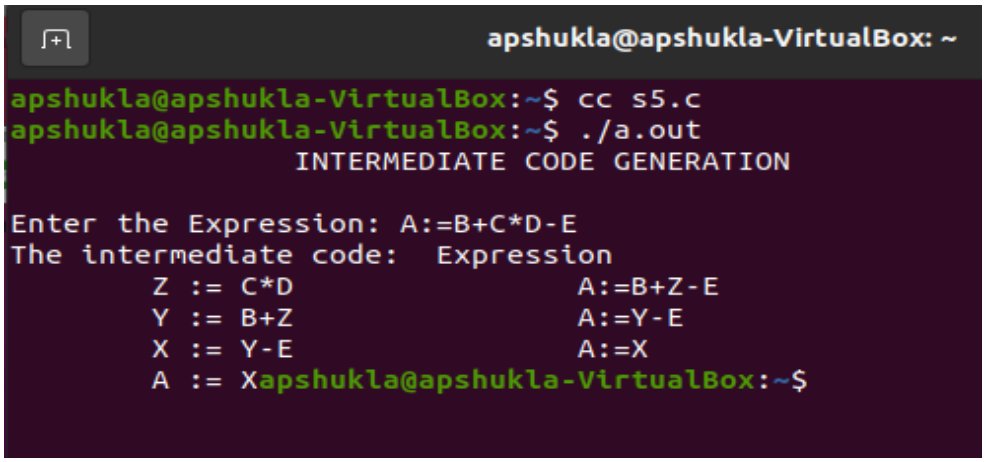
```
 int pos;
 char op;
}k[15];
void main()
{

 printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
 printf("Enter the Expression :");
 scanf("%s",str);
 printf("The intermediate code:\t\tExpression\n");
 findopr();
 explore();

}
void findopr()
{
 for(i=0;str[i]!='\0';i++)
  if(str[i]==':')
  {
  k[j].pos=i;
  k[j++].op=':';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='/')
  {
  k[j].pos=i;
  k[j++].op='/';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='*')
```

```
  {
  k[j].pos=i;
  k[j++].op='*';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='+')
  {
  k[j].pos=i;
  k[j++].op='+';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='-')
  {
  k[j].pos=i;
  k[j++].op='-';
  }
}
void explore()
{
 i=1;
 while(k[i].op!='\0')
 {
  fleft(k[i].pos);
  fright(k[i].pos);
  str[k[i].pos]=tmpch--;
  printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
  for(j=0;j <strlen(str);j++)
   if(str[j]!='$')
    printf("%c",str[j]);
  printf("\n");
```

```
 i++;
 }
 fright(-1);
 if(no==0)
 {
  fleft(strlen(str));
  printf("\t%s := %s",right,left);

  exit(0);
 }
 printf("\t%s :=  %c",right,str[k[--i].pos]);

}
void fleft(int x)
{
 int w=0,flag=0;
 x--;
 while(x!= -1 &&str[x]!= '+' &&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-
'&&str[x]!='/'&&str[x]!=':')
 {
  if(str[x]!='$'&& flag==0)
  {
  left[w++]=str[x];
  left[w]='\0';
  str[x]='$';
  flag=1;
  }
  x--;
 }
}
void fright(int x)
```

```
{
 int w=0,flag=0;
 x++;
 while(x!= -1 && str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')
 {
  if(str[x]!='$'&& flag==0)
  {
  right[w++]=str[x];
  right[w]='\0';
  str[x]='$';
  flag=1;
  }
  x++;
 }
}
```

Output: -



6. a. Write a LEX program to eliminate *comment lines* in a *C* program and copy the resulting program into a separate file.

Lex Code: -

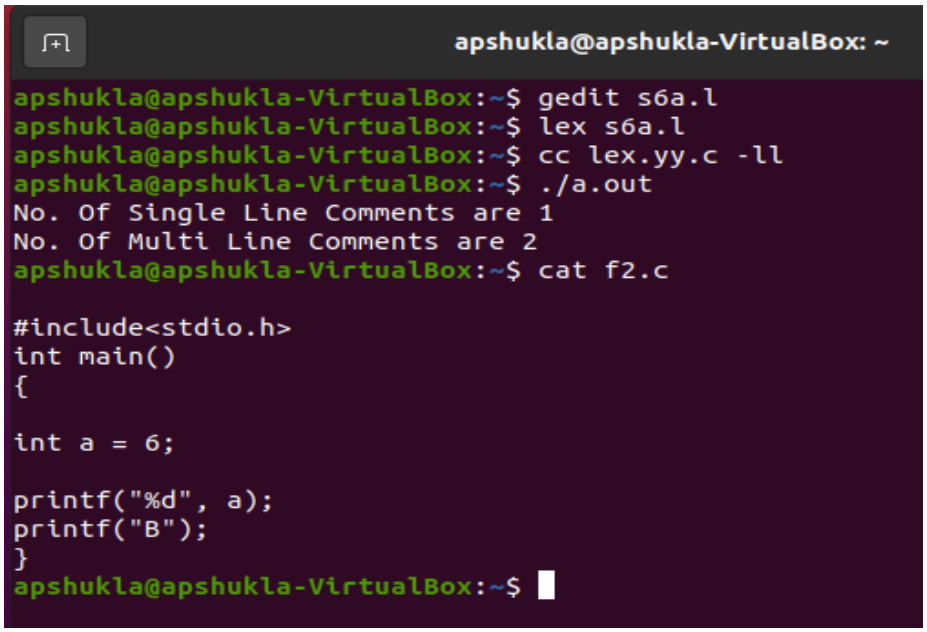
```
%{
int sl=0;
int ml=0;
```

```
%}
%%
"//".*    { sl++; }
"/*"[a-zA-Z0-9' '\t\n\(\)]+"*/" { ml++; }
%%
int main()
{
yyin=fopen("f1.c","r");
yyout=fopen("f2.c","w");
yylex();
printf("No. Of Single Line Comments are %d \n", sl);
printf("No. Of Multi Line Comments are %d \n", ml);
fclose(yyin);
fclose(yyout);
}
```

C Prog (f1.c): -

```
/*6a program for printing
number of single and multi line comments*/
#include<stdio.h>
int main()
{
/*6th Sem CSE*/
int a = 6;
//print value of a
printf("%d", a);
printf("B");
}
```

Output: -



6. b. Write YACC program to recognize valid *identifier, operators and keywords* in the given text
(*C program*) file.

Lex Code: -

```
%{
#include<stdio.h>
#include"y.tab.h"
extern int yylval;
%}

%%
[\t];
[+|-|*|/|=|<|>]   { printf("Operator is %s \n", yytext); return OP; }
[0-9]+                  { yylval=atoi(yytext); printf("Number is %s \n", yytext); return DIGIT; }
int|char|bool|include|main|printf|float|void|for|do|while|if|else|return { printf("Keyword is %s \n", yytext); return KEY; }
[a-zA-Z][a-zA-Z0-9]* { printf("Identifier is %s \n", yytext); return ID; }
```

```
. ;
%%
```

Yacc Code: -

```
%{
#include<stdio.h>
#include<stdlib.h>
int id=0, dig=0, key=0, op=0;
extern int yylex();
void yyerror();
extern int yyparse();
%}
%token DIGIT ID KEY OP

%%
input:
DIGIT input    { dig++; }
|ID input      { id++; }
|KEY input     { key++; }
|OP input      { op++; }
|DIGIT         {dig++; }
|ID            { id++; }
|KEY           { key++; }
|OP            { op++; }
;
%%

extern FILE *yyin;
int main()
{
        FILE *myfile = fopen("f3.c", "r");
```

```
        if(!myfile)
        {
                printf("I can't open f3.c!");
                return -1;
        }
        yyin = myfile;
        do{
                yyparse();
        }while(!feof(yyin));
        printf("Numbers=%d \nKeywords=%d \nIdentifiers=%d \nOperators= %d \n", dig,
key, id, op);
}
void yyerror()
{
        printf("Parse error! message: ");
        exit(-1);
}
```

C Prog (f3.c): -

```
#include<stdio.h>
int a, b, c=5;
float sum;
char str;
int main()
{
   d=10;
   sum=5.5;
   printf("The sum = %d", sum);
}
```

Output: -

```
apshukla@apshukla-VirtualBox:~$ lex s6b.l
apshukla@apshukla-VirtualBox:~$ yacc -d s6b.y
apshukla@apshukla-VirtualBox:~$ cc lex.yy.c y.tab.c -ll
apshukla@apshukla-VirtualBox:~$ ./a.out
Keyword is include
Operator is <
Identifier is stdio
Identifier is h
Operator is >

Keyword is int
Identifier is a
Identifier is b
Identifier is c
Operator is =
Number is 5

Keyword is float
Identifier is sum

Keyword is char
Identifier is str

Keyword is int
Keyword is main


Identifier is d
Operator is =
Number is 10

Identifier is sum
Operator is =
Number is 5
Number is 5

Keyword is printf
Identifier is The
Identifier is sum
Operator is =
Identifier is d
Identifier is sum


Numbers=4
Keywords=7
Identifiers=13
Operators= 6
apshukla@apshukla-VirtualBox:~$
```

7. Design, develop and implement a C/C++/Java program to simulate the working of Shortest remaining time and Round Robin (RR) scheduling algorithms. Experiment with different quantum sizes for RR algorithm.

```
#include<stdio.h>

#include<stdlib.h>

//#include<conio.h>

struct proc

{

int id;

int arrival;

int burst;
```

```
int rem;
int wait;
int finish;
int turnaround;
float ratio;
}process[10]; //structure to hold the process information
struct proc temp;
int no;
int chkprocess(int);
int nextprocess();
void roundrobin(int, int, int[], int[]);
void srtf(int);

int main()
{
int n,tq,choice;
int bt[10],st[10],i,j,k;
for(; ;)
{
printf("Enter the choice \n");
printf(" 1. Round Robin\n 2. SRT\n 3. Exit \n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Round Robin scheduling algorithm\n");
printf("Enter number of processes:\n");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
```

```
{
scanf("%d",&bt[i]);
st[i]=bt[i]; //service time
}
printf("Enter time quantum:");
scanf("%d",&tq);
roundrobin(n,tq,st,bt);
break;
case 2:
printf("\n \n ---SHORTEST REMAINING TIME NEXT---\n \n ");
printf("\n \n Enter the number of processes: ");
scanf("%d", &n);
srtf(n);
break;
case 3:
exit(0);
}// end of switch
}// end of for
}//end of main()
void roundrobin(int n,int tq,int st[],int bt[])
{
int time=0;
int tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,sq=0,j,k;
float awt=0.0,atat=0.0;
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp1=tq;
if(st[i]==0) // when service time of a process equals zero then
```

```
//count value is incremented
{
count++;
continue;
}
if(st[i]>tq) // when service time of a process greater than time
//quantum then time
st[i]=st[i]-tq; //quantum value subtracted from service time
else
if(st[i]>=0)
{
temp1=st[i]; // temp1 stores the service time of a process
st[i]=0; // making service time equals 0
}
sq=sq+temp1; // utilizing temp1 value to calculate turnaround time
tat[i]=sq; // turn around time
} //end of for

if(n==count) // it indicates all processes have completed their task because the count value
break; // incremented when service time equals 0
} //end of while
for(i=0;i<n;i++) // to calculate the wait time and turnaround time of each process
{ wt[i]=tat[i]-bt[i]; // waiting time calculated from the turnaround time burst time
swt=swt+wt[i]; // summation of wait time
stat=stat+tat[i]; // summation of turnaround time
}
awt=(float)swt/n; // average wait time
atat=(float)stat/n; // average turnaround time
printf("Process_no Burst time \tWait time Turn around time\n");
for(i=0;i<n;i++)
```

```
printf("%d\t\t%d\t\t%d\t\t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("Avg wait time is %f\n Avg turn around time is %f\n",awt,atat);
}// end of Round Robin
int chkprocess(int s) // function to check process remaining time is zero or not
{
int i;
for(i = 1; i <= s; i++)
{
if(process[i].rem != 0)
return 1;
}
return 0;
} // end of chkprocess
int nextprocess() // function to identify the next process to be executed
{
int min, l, i;
min = 32000; //any limit assumed
for(i = 1; i <= no; i++)
{
if( process[i].rem!=0 && process[i].rem < min)
{
min = process[i].rem;
l = i;
}
}
return l;
} // end of nextprocess
void srtf(int n)
{
int i,j,k,time=0;
```

```
float tavg,wavg;
for(i = 1; i <= n; i++)
{
process[i].id = i;
printf("\n\nEnter the arrival time for process %d: ", i);
scanf("%d", &(process[i].arrival));
printf("Enter the burst time for process %d: ", i);
scanf("%d", &(process[i].burst));
process[i].rem = process[i].burst;
}
for(i = 1; i <= n; i++)
{
for(j = i + 1; j <= n; j++)
{
if(process[i].arrival > process[j].arrival)
// sort arrival time of a process
{
temp = process[i];
process[i] = process[j];
process[j] = temp;
}
}
}
no = 0;
j = 1;
while(chkprocess(n) == 1)
{
if(process[no + 1].arrival == time)
{
while(process[no+1].arrival==time)
```

```
no++;
if(process[j].rem==0)
process[j].finish=time;
j = nextprocess();
}
if(process[j].rem != 0) // to calculate the waiting time of aprocess
{
process[j].rem--;
for(i = 1; i <= no; i++)
{
if(i != j && process[i].rem != 0)
process[i].wait++;
}
}
else
{
process[j].finish = time;
j=nextprocess();
time--;
k=j;
}
time++;
}
process[k].finish = time;
printf("\n\n\t\t\t---SHORTEST REMAINING TIME FIRST---");
printf("\n\n Process \tArrival Burst Waiting Finishing turnaround Tr/Tb\n");
printf("%5s %9s %7s %10s %8s %9s\n\n", "id", "time", "time", "time","time", "time");
for(i = 1; i <= n; i++)
{
process[i].turnaround = process[i].wait + process[i].burst; // calc of turnaround
```

```
process[i].ratio = (float)process[i].turnaround /(float)process[i].burst;

printf("%5d %8d %7d %8d %10d %9d %10.1f ", process[i].id,process[i].arrival,
process[i].burst,

process[i].wait, process[i].finish,process[i].turnaround, process[i].ratio);

tavg=tavg+ process[i].turnaround; //summation of turnaround time

wavg=wavg+process[i].wait; // summation of waiting time

printf("\n\n");

}

tavg=tavg/n; // average turnaround time

wavg=wavg/n; // average wait time

printf("tavg=%f\t wavg=%f\n",tavg,wavg); }// end of srtf
```

Output: -

```
apshukla@apshukla-VirtualBox:~$ gedit s7.c
apshukla@apshukla-VirtualBox:~$ cc s7.c
apshukla@apshukla-VirtualBox:~$ ./a.out
Enter the choice
 1. Round Robin
 2. SRT
 3. Exit
1
Round Robin scheduling algorithm
Enter number of processes:
6
Enter burst time for sequences:4
5
2
1
6
3
Enter time quantum:2
Process_no Burst time    Wait time Turn around time
1                4                9                13
2                5                14               19
3                2                4                6
4                1                6                7
5                6                15               21
6                3                15               18
Avg wait time is 10.500000
 Avg turn around time is 14.000000
Enter the choice
 1. Round Robin
 2. SRT
 3. Exit
```

```
Enter the choice
 1. Round Robin
 2. SRT
 3. Exit
2


 ---SHORTEST REMAINING TIME NEXT---



 Enter the number of processes: 6


Enter the arrival time for process 1: 0
Enter the burst time for process 1: 7


Enter the arrival time for process 2: 1
Enter the burst time for process 2: 5


Enter the arrival time for process 3: 2
Enter the burst time for process 3: 3


Enter the arrival time for process 4: 3
Enter the burst time for process 4: 1


Enter the arrival time for process 5: 4
Enter the burst time for process 5: 2


Enter the arrival time for process 6: 5
Enter the burst time for process 6: 1
```

```
Enter the arrival time for process 4: 3
Enter the burst time for process 4: 1


Enter the arrival time for process 5: 4
Enter the burst time for process 5: 2


Enter the arrival time for process 6: 5
Enter the burst time for process 6: 1


                    ---SHORTEST REMAINING TIME FIRST---

 Process        Arrival Burst Waiting Finishing turnaround Tr/Tb
   id      time    time       time     time      time

    1        0       7       24         19        31        4.4

    2        1       5       14         13        19        3.8

    3        2       3        2          6         5        1.7

    4        3       1        0          4         1        1.0

    5        4       2        6          9         8        4.0

    6        5       1        2          7         3        3.0

tavg=-6205930.000000     wavg=8.000000
Enter the choice
 1. Round Robin
 2. SRT
 3. Exit
3
apshukla@apshukla-VirtualBox:~$
```

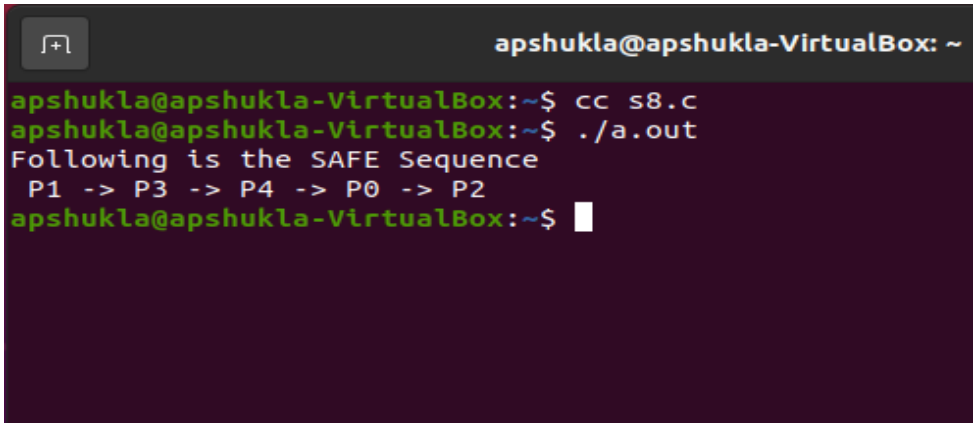
8. Design, develop and implement a C/C++/Java program to implement Banker's algorithm.
Assume suitable input required to demonstrate the results.

```
#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k;

n=5; // Number of processes

m=3; // Number of resources

int alloc[5][3] = { { 0, 1, 0 }, { 2, 0, 0 }, { 3, 0, 2 }, { 2, 1, 1 }, { 0, 0, 2 } };

int max[5][3] = { {7, 5,3 },{ 3,2,2}, {9,0,2},{2,2,2}, {4,3,3,}};

int avail[3] = { 3, 3, 2 }; // Available Resources

int f[n], ans[n], ind = 0;
```

```
for (k = 0; k < n; k++) {
f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
for (i = 0; i < n; i++) {
if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){
flag = 1;
break;
}
}
if (flag == 0) {
ans[ind++] = i;
for (y = 0; y < m; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
```

```
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);
printf("\n");
return (0);
// This code is contributed by Deep Baldha (CandyZack)
}
```

Output: -



9. Design, develop and implement a C/C++/Java program to implement page replacement algorithms LRU and FIFO. Assume suitable input required to demonstrate the results.

```
#include<stdio.h>
#include<stdlib.h>
void FIFO (char[], char[], int, int);
void lru (char[], char[], int, int);
int main()
{
  int ch, YN = 1, i, l, f;
  char F[10], s[25];
  printf("\n\t Enter the no of empty frames: ");
  scanf("%d", &f);
  printf("\n\t Enter the length of the string: ");
  scanf("%d", &l);
  printf("\n\t Enter the string: ");
  scanf("%s", &s);
```

```
 for(i = 0; i < f; i++)
        F[i] = -1;

 do
 {
        printf("\n\t *******MENU*******");
        printf("\n\t 1.FIFO\n\n\t 2.LRU\n\n\t 3.Exit");
        printf("\n Enter your choice");
        scanf("%d", &ch);
        switch(ch)
        {
                case 1:
                        for(i=0; i<f; i++)
                        {
                          F[i] = -1;
                        }
                        FIFO(s, F, l, f);
                        break;
                case 2:
                        for(i=0; i<f; i++)
                        {
                          F[i] = -1;
                        }
                        lru(s, F, l, f);
                        break;
                case 3:
                        exit(0);
        }
        printf("\n\n\t DO you want to continue IF YES PRESS 1 \n\n\t IF NO PRESS 0 : ");
        scanf("%d", &YN);
```

```
  } while(YN == 1);
  return 0;
}

void FIFO (char s[], char F[], int l, int f)
{
  int i, j=0, k, flag=0, cnt=0;
  printf("\n\t PAGE \t FRAMES \t FAULTS");
  for(i=0; i<l; i++)
  {
    for(k=0; k<f; k++)
    {
      if(F[k] == s[i])
            flag = 1;
    }
    if(flag == 0)
    {
      printf("\n\t %c \t", s[i]);
      F[j] = s[i];
      j++;
      for(k=0; k<f; k++)
      {
        printf(" %c", F[k]);
      }
      printf("\t Page-fault%d", cnt);
      cnt++;
    }
    else
    {
      flag = 0;
```

```
      printf("\n\t %c \t", s[i]);
      for(k=0; k<f; k++)
      {
        printf(" %c", F[k]);
      }
      printf("\t No Page-fault");
    }
    if(j == f)
          j=0;
  }
}

void lru(char s[], char F[], int l, int f)
{
  int i, j=0, k, m, flag=0, cnt=0, top=0;
  printf("\n\t PAGE \t FRAMES \t FAULTS");
  for(i=0; i<l; i++)
  {
    for(k=0; k<f; k++)
    {
      if(F[k] == s[i])
      {
        flag=1;
        break;
      }
    }
    printf("\n\t %c \t", s[i]);
    if(j != f && flag != 1)
    {
      F[top] = s[i];
```

```
  j++;
  if(j!=f)
       top++;
 }
 else
 {
  if(flag != 1)
  {
    for(k=0; k<top; k++)
    {
      F[k] = F[k+1];
    }
    F[top] = s[i];
  }
  if(flag == 1)
  {
    for(m=k; m<top; m++)
    {
      F[m] = F[m+1];
    }
    F[top] = s[i];
  }
 }
 for(k=0; k<f; k++)
 {
  printf(" %c", F[k]);
 }
 if(flag == 0)
 {
  printf("\t Page-fault%d", cnt);
```

```
      cnt++;
    }
    else
          printf("\t No page-fault");
    flag=0;
  }
}
```

Output: -

```
apshukla@apshukla-VirtualBox:~$ ./a.out

         Enter the no of empty frames: 4

         Enter the length of the string: 10

         Enter the string: 2342137543

         *******MENU*******
         1.FIFO

         2.LRU

         3.Exit
 Enter your choice1

         PAGE    FRAMES          FAULTS
         2       2 ◆ ◆ ◆         Page-fault0
         3       2 3 ◆ ◆         Page-fault1
         4       2 3 4 ◆         Page-fault2
         2       2 3 4 ◆         No Page-fault
         1       2 3 4 1         Page-fault3
         3       2 3 4 1         No Page-fault
         7       7 3 4 1         Page-fault4
         5       7 5 4 1         Page-fault5
         4       7 5 4 1         No Page-fault
         3       7 5 3 1         Page-fault6

         DO you want to continue IF YES PRESS 1

         IF NO PRESS 0 : 1
```

```
        DO you want to continue IF YES PRESS 1

        IF NO PRESS 0 : 1

        *******MENU*******
        1.FIFO

        2.LRU

        3.Exit
 Enter your choice2

        PAGE    FRAMES          FAULTS
        2       2 ◆ ◆ ◆         Page-fault0
        3       2 3 ◆ ◆         Page-fault1
        4       2 3 4 ◆         Page-fault2
        2       3 4 ◆ 2         No page-fault
        1       3 4 ◆ 1         Page-fault3
        3       4 ◆ 1 3         No page-fault
        7       ◆ 1 3 7         Page-fault4
        5       1 3 7 5         Page-fault5
        4       3 7 5 4         Page-fault6
        3       7 5 4 3         No page-fault

        DO you want to continue IF YES PRESS 1

        IF NO PRESS 0 : 0
apshukla@apshukla-VirtualBox:~$
```