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.

```
%{

#include<stdio.h> int
v=0,op=0,id=0;
%}
%%
[0-9]+ {id++; printf("\n identifers ");ECHO;}[\+ \*]
{op++; printf("\n operators");ECHO;}"("  {v++;}
")" {v--;}
.|\n {;}
%%
int main()
{
printf("Enter the expression");
yylex();
if((op+1)==id && v==0)
printf("Expression is valid \n");else
printf("Expression is invalid \n");
printf("No of identifers=%d",id);
printf("No of operators=%d",op);
}
```

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

Lex part

```
%{
#include "y.tab.h"extern
yylval;
%}
%%
[0-9]+ {yylval=atoi(yytext); return num;}[\+\-
\*\/] {return yytext[0];}
[)] {return yytext[0];}
[(] {return yytext[0];}
. {;}
\n {return 0;}
%%
```

Yacc part

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%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("Divide by zero \n");exit(0);}else
        $$=$1/$3;}
        |'(' exp ')' {$$=$2;}
        |num{$$=$1;}
        ;
        %%
        int yyerror()
        {
printf("error");
exit(0);
        }
        int main()
        {
printf("Enter an expression \n");
yyparse();
        }
```

Output

Lex 1b.l Yacc –d

1b.y

Cc –lm y.tab.c lex.yy.c

2 a) Develop, Implement and Execute a program using YACC tool to recognize all strings ending with *b* preceded by *n a's* using the grammar $a^n b$

lex part

```
        %{
        #include "y.tab.h"
        %}
        %%
a {return A;} b
{return B;} [\n]
return '\n';
```

```
%%
```

Yacc

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B
%%
input:S'\n' {printf("Successfull grammar \n");exit(0);}S:A A
S B S1|;
S1:B B|;
%%
int main()
{
printf("Enter a string \n");
yyparse();
}
int yyerror()
{
printf("Error \n");
exit(0);
}
```

3a) 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<string.h>int
main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,f,i,t,b,c,n,k,l=0,h,j,s,m,p,q2;
printf("enter the number of production");
scanf("%d",&n)
printf("enter producive grammer");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++)
{
for(i=0;i<n;i++)
{ j=3;l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
```

```
        {
        for(m=0;m<l;m++)
        {
if(ft[i][m]==st[i][j])goto
s1;
        }
ft[i][1]=st[i][j];l=l+1;
s1:j=j+1;
        }
        else
        {
        if(s>0)
        {
        while(st[i][j]!=st[a][0])
{ a++;
} b=0;
        while(ft[a][b]!='\0')
        {
        for(m=0;m<l;m++)
        {
if(ft[i][m]==ft[a][b])goto
s2;
        }
ft[i][1]=ft[a][b];l=l+1;
s2:b=b+1;
        }
        }
        }
        while(st[i][j]!='\0')
        {
        if(st[i][j]=='|')
        {
j=j+1; goto l1;
        }
        j=j+1;
} ft[i][1]='\0';
        }
        }
printf("first pos");
for(i=0;i<n;i++)
printf("first[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
        for(i=0;i<n;i++)
        {
```

```
k=0;j=3;
if(i==0)l=1;
elsel=0;
        k1:while((st[i][0]!=st[k][j])&&(k<n))
        {
        if(st[k][j]=='\0')
{ k++;j=2;
} j++;
        }
j=j+1; if(st[i][0]==st[k][j-1])
        {
        if((st[k][j]!='|')&&(st[k][j]!='\0'))
{ a=0;
        if(!((st[k][j]>64)&&(st[k][j]<91)))
        {
        for(m=0;m<l;m++)
        {
if(fol[i][m]==st[k][j])goto
q3;
        }
q3: fol[i][l]=st[k][j];
l++;
        }
        else
        {
        while(st[k][j]!=st[a][0])
{ a++;
} p=0;
        while(ft[a][p]!='\0')
        {
        if(ft[a][p]!='@')
        {
        for(m=0;m<l;m++)
        {
if(fol[i][m]==ft[a][p])goto
q2;
        }
fol[i][1]=ft[a][p];l=l+1;
```

```
        }
else e=1;
q2:p++;
        }
        if(e==1)
{ e=0;
        goto a1;
        }
        }
        }
        else
{ a1:c=0;a=0;
        while(st[k][0]!=st[a][0])
{ a++;
        }
        while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
        {
        for(m=0;m<l;m++)
        {
if(fol[i][m]==fol[a][c])goto
q1;
        }
fol[i][1]=fol[a][c];l++;
        q1:c++;
        }
        }
        goto k1;
        }
        fol[i][1]='\0';
        }
printf("follow pos");
for(i=0;i<n;i++)
printf("follow[%c]=%s",st[i][0],fol[i]);
printf("\n");
        s=0;
        for(i=0;i<n;i++)
{ j=3;
        while(st[i][j]!='\0')
        {
        if((st[i][j-1]=='|')||(j==3))
        {
        for(p=0;p<=2;p++)
        {
```

```
        fin[s][p]=st[i][p];
        }
t=j; for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)
        {
fin[s][p]=st[i][j];j++;
        }
        fin[s][p]='\0';
        if(st[i][t]=='@')
{ b=0;a=0;
        while(st[a][0]!=st[i][0])
{ a++;
        }
        while(fol[a][b]!='\0')
        {
printf("M[%c%c]=%s\n",st[i][0],fol[a][b],fin[s]);b++;
        }
        }
        else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);else
{ b=0;a=0;
        while(st[a][0]!=st[i][3])
{ a++;
        }
        while(ft[a][b]!='\0')
        {
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);b++;
        }
} s++;
        }
if(st[i][j]=='|')j++;
        }
        }

        }
```

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<stdlib.h>
#include<string.h>
char    ip_sym[15],stack[15];   int
ip_ptr=0,st_ptr=0,len,i;        char
temp[2];
char   act[15];  void
check();void main()
        {
printf("\n\t\t SHIFT REDUCE PARSER \n");
printf("\n Grammer\n");
printf("E->E+T|T\nT->T*F|F\nF->(E)|id\n");
printf("enter the input symbol:\t"); scanf("%s",ip_sym);
printf("\n\tstack implimentation table\n");
printf("\nStack\t\tinput symbol\t\taction");
printf("\n$\t\t%s$\t\t--",ip_sym);
strcpy(act,"shift");
        if(ip_sym[ip_ptr]=='(')
        {
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
        }
        else
        {
temp[0]=ip_sym[ip_ptr];
temp[1]=ip_sym[ip_ptr+1];
temp[1]='\0';
        }
strcat(act,temp);
len=strlen(ip_sym);
for(i=0;i<i<=len-1;i++)
        {
        if(ip_sym[ip_ptr]=='i'&&ip_sym[ip_ptr+1]=='d')
        {
stack[st_ptr]=ip_sym[ip_ptr];
st_ptr++;
ip_sym[ip_ptr]=' ';
ip_ptr++;
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
        ip_ptr++;
        }
```

```
        else
        {
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
        ip_ptr++;
        }
printf("\n$%s\t\t%s$\t\t%s",stack,ip_sym,act);
strcpy(act,"shift");
        if(ip_sym[ip_ptr]=='('||ip_sym[ip_ptr]=='*'||ip_sym[ip_ptr]=='+'||ip_sym[ip_ptr]==')')
        {
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
        }
        else
        {
temp[0]=ip_sym[ip_ptr];
temp[1]=ip_sym[ip_ptr+1];
temp[2]='\0';
        }
strcat(act,temp);
check(); st_ptr++;
        }
st_ptr++;
check();
        }
        void check()
        {
int flag=0;
while(1)
        {
        if(stack[st_ptr]=='d'&&stack[st_ptr-1]=='i')
        {
stack[st_ptr-1]='F';
stack[st_ptr]=='\0'; st_ptr-
-;
        flag=1;
        printf("\n$%s\t\t%s$\t\tF->id",stack,ip_sym);
        }
if(stack[st_ptr]==')'&&stack[st_ptr-1]=='E'&&stack[st_ptr-2]=='('){
stack[st_ptr-2]='F';
stack[st_ptr-1]='\0';flag=1;
st_ptr=st_ptr-2;
        printf("\n$%s\t\t%s$\t\tF->id",stack,ip_sym);
        }
        if(stack[st_ptr]=='F'&&stack[st_ptr-1]=='*'&&stack[st_ptr-2]=='T')
        {
        stack[st_ptr-1]='\0';
```

```
st_ptr=st_ptr-2;
flag=1;
        printf("\n$%s\t\t%s$\t\tF->T*F",stack,ip_sym);
        }
        else
        {
        if(stack[st_ptr]=='F')
        {
stack[st_ptr]='T';flag=1;
        printf("\n$%s\t\t%s$\t\tT->F",stack,ip_sym);
        }
        }
        if(stack[st_ptr]=='T'&&stack[st_ptr-1]=='+'&&stack[st_ptr-2]=='E'&&ip_sym[ip_ptr]!='*')
        {
stack[st_ptr-1]='\0';
st_ptr=st_ptr-2; flag=1;
        printf("\n$%s\t\t%s$\t\tE->E+T",stack,ip_sym);
        }
else if((stack[st_ptr]=='T'&&ip_sym[ip_ptr]=='+')||
(stack[0]=='T'&&ip_sym[ip_ptr]=='\0')||
(stack[st_ptr]=='T'&&ip_sym[ip_ptr]==')'))
        {
stack[st_ptr]='E';flag=1;
        printf("\n$%s\t\t%s$\t\tE->T",stack,ip_sym);
        }
if((stack[st_ptr]=='T'&&ip_sym[ip_ptr]=='*')||
(stack[st_ptr]=='E'&&ip_sym[ip_ptr]==')')||
(stack[st_ptr]=='E'&&ip_sym[ip_ptr]=='+')||
(stack[st_ptr]=='+'&&ip_sym[ip_ptr]=='i'&&ip_sym[ip_ptr+1]=='d')||
(stack[st_ptr]=='('&&ip_sym[ip_ptr]=='i'&&ip_sym[ip_ptr+1]=='d')||
(stack[st_ptr]=='*'&&ip_sym[ip_ptr]=='i'&&ip_sym[ip_ptr+1]=='d')||
(stack[st_ptr]=='('&&ip_sym[ip_ptr]=='(')||
(stack[st_ptr]=='*'&&ip_sym[ip_ptr]=='(')||
(stack[st_ptr]=='+'&&ip_sym[ip_ptr]=='(')
        {
        flag=2;
        }
        if(!strcmp(stack,"E")&&ip_sym[ip_ptr]=='\0')
        {
        printf("\n$%s\t\t%s$\t\tAccept",stack,ip_sym);
//getch();
exit(0);
        }
        if(flag==0)
        {
        printf("\n%s\t\t\t%s\treject",stack,ip_sym);
```

```
        exit(0);
        }
if(flag==2)return;
flag=0;
        }
        }
```

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
T2 = C + D
T3 = T1 + T2
A = T3

```
#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

char op[2],arg1[5],arg2[5],result[5];void

main()

        {

FILE *fp1,*fp2;

fp1=fopen("input.txt","r");

fp2=fopen("output.txt","w");

while(!feof(fp1))


        {

fscanf(fp1,"%s%s%s%s",result,arg1,op,arg2);

if(strcmp(op,"+")==0)

        {

        fprintf(fp2,"\nMOV R0,%s",arg1);


fprintf(fp2,"\nADD R0,%s",arg2);

fprintf(fp2,"\nMOV %s,R0",result);
```

```
        }
        if(strcmp(op,"*")==0)
        {

fprintf(fp2,"\nMOV         R0,%s",arg1);
fprintf(fp2,"\nMUL         R0,%s",arg2);
fprintf(fp2,"\nMOV %s,R0",result);
        }

        if(strcmp(op,"-")==0)
        {
fprintf(fp2,"\nMOV R0,%s",arg1);
fprintf(fp2,"\nSUB R0,%s",arg2);
fprintf(fp2,"\nMOV %s,R0",result);

        }
        if(strcmp(op,"/")==0)
        {
fprintf(fp2,"\nMOV R0,%s",arg1);
fprintf(fp2,"\nDIV R0,%s",arg2);

        fprintf(fp2,"\nMOV %s,R0",result);
        }
        if(strcmp(op,"=")==0)
        {

fprintf(fp2,"\nMOV R0,%s",arg1);
fprintf(fp2,"\nMOV %s,R0",result);
        }
```

```
fclose(fp1);

fclose(fp2);

getch();

        }
```

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

```
        %{
#include<stdlib.h>int
c_count=0;
        %}
        %%
"/*"[^*/]*"*/" {c_count++;}"//".*
{c_count++;}
        %%
        int main(int argc,char **argv)
        {
FILE *f1,*f2;
if(argc>1)
        {
f1=fopen(argv[1],"r");if(!f1)
        {
printf("file error \n");
exit(1);
        }
yyin=f1; f2=fopen(argv[2],"w");
if(!f2)
        {
printf("Error");
exit(1);
        }
yyout=f2;
yylex();
        printf("number of comment lines:%d \n",c_count);
        }
        return 0;
        }
```

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

Lex part

```
%{
#include<stdio.h>
#include "y.tab.h" extern
yylval;
%}
%%
[\t];
[+|-|*|/|=|<|>] {printf("operator is %s \n",yytext);return OP;}
[0-9]+ {yylval=atoi(yytext); printf("numbers is %d \n",yylval);return DIGIT;}
int|char|bool|float|void|for|do|while|if|else|return|void {printf("keyword is %s
\n",yytext);return KEY;}
[a-zA-Z0-9]+ {printf("identifers is %s \n",yytext);return ID;}
. ;
%%
```

Yacc part

```
%{
#include<stdio.h>
#include<stdlib.h>
int id=0,dig=0,key=0,op=0;
%}
%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++;}
;
%%
#include<stdio.h> extern
int yylex(); extern int
yyparse();extern FILE
*yyin; main()
{
FILE *myfile=fopen("sam_input.c","r");
if(!myfile) {
printf(" I can't open sam_input.c!");return
-1;
}
yyin=myfile;
```

```
do{ yyparse();
        }
        while(!feof(yyin));
        printf("numbers=%d\nKeywords=%d\nIdentifers=%d\noperators=%d\n",dig,key,id,op);
        }
        void yyerror()
        {
printf("EEK,parse error!Message:");exit(-
1);
        }
```

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>struct
proc
        {

        int id;
int arrival;int
burst; int rem; int
wait;

        int finish;
int turnaround;float
ratio;
        }process[10];

struct proc temp;int no;
int chkprocess(int);int
nextprocess();
        void roundrobin(int, int, int[], int[]);

void srtf(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          Wait 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 assumedfor(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 a process
        {
        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 Arrival 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; 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
```

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>
#include <stdlib.h>
int Max[10][10], need[10][10], alloc[10][10], avail[10], finished[10],
safeSequence[10],work[10],request[10][10];
int p, r, i,rp,j,k, process, count,vc,wc;int
found=0;

        int safestate()
        {
                count=0;
                for(i=0;i<p;i++)
                        finished[i]=-1;
                        for (j=0;j < r;j++)
                                work[j]=avail[j];

                while(count!=p)
                        {
                                found=0;
                                for(i=0;i<p;i++)
                                {
                                        if(finished[i]==-1)
                                        {
                                                for(j=0;j<r;j++)
                                                {
                                                        if(need[i][j]>work[j])
                                                                break;
                                                }
```

```
                                        if(j==r)
                                        {
                                                for(k=0;k<r;k++)
                                                        work[k]=work[k]+alloc[i][k];
                                                safeSequence[count++]=i;
                                                finished[i]=1;
                                printf("process %d finished \n",i);
                                                found=1;
                                        }

                                }

                                }
                                if(found==0)
                                {
                        printf("\nThe system is in an unsafe state!!");
                                break;}
}
                                if(count == p)

                                {

                                printf("\nThe system is in a safe state!!\n"); printf("Safe Sequence : < ");
                                for( i = 0; i < p; i++)
                                        printf("%d ", safeSequence[i]);

                                printf(">\n");
                                }

}

int main()
{
        printf("Enter the no of processes : ");
      scanf("%d", &p);
      printf("\n\nEnter the no of resources : "); scanf("%d", &r);
      printf("\n\nEnter the Max Matrix for each process : ");
      for(i = 0; i < p; i++)

      {
            printf("\nFor process %d : ", i + 1);
            for(j = 0; j < r; j++)
                  scanf("%d", &Max[i][j]);
      }

      printf("\n\nEnter the allocation for each process : ");
      for(i = 0; i < p; i++)
      {
            printf("\nFor process %d : ",i + 1);
            for(j = 0; j < r; j++)
```

```
        scanf("%d", &alloc[i][j]);
 }

 printf("\n\nEnter the Available Resources : ");
 for(i = 0; i < r; i++)
        scanf("%d", &avail[i]);
 for(i = 0; i < p; i++)
        for(j = 0; j < r; j++)
                need[i][j] = Max[i][j] - alloc[i][j];


printf("\n Max matrix:\tAllocation matrix:\tNeed Matrix\n");
                    for(i = 0; i < p; i++)
        {

                for( j = 0; j < r; j++)
                        printf("%d ", Max[i][j]);
                printf("\t\t");
                for( j = 0; j < r; j++)
                        printf("%d ", alloc[i][j]);
                printf("\t\t\t");
                for( j = 0; j < r; j++)
                        printf("%d ", need[i][j]);

                printf("\n");
        }
                safestate();
      vc=0,wc=0;
 printf("\n Requesting process id:");
 scanf("%d",&rp);
 printf("enter the resources for requested process\n");
 for(j=0;j<r;j++)
 {
        scanf("%d",&request[rp][j]);
        if(request[rp][j]>need[rp][j])
                {
                vc=1;
                printf("request[%d]>need[%d]",j,j);
                }
        if(request[rp][j]>avail[j])
                {
                wc=1;
                printf("request[%d]>avail[%d]",j,j);
                }
 }
 if(vc==1)
 printf("error\n");
 else if(wc==1)
 printf("error\n");
 else
```

```
        {
            for(j=0;j<r;j++)
            {
                avail[j]=avail[j]-request[rp][j];
                alloc[rp][j]=alloc[rp][j]+request[rp][j];
                need[rp][j]=need[rp][j]-request[rp][j];
            }
        printf("Resource request commited\n");
          printf("Updated Matrices\n");
          printf("\n Max matrix:\tAllocation matrix:\tNeed Matrix\n");
                    for(i = 0; i < p; i++)
            {

                for( j = 0; j < r; j++)
                    printf("%d ", Max[i][j]);
                printf("\t\t");
                for( j = 0; j < r; j++)
                    printf("%d ", alloc[i][j]);
                printf("\t\t\t");
                for( j = 0; j < r; j++)
                    printf("%d ", need[i][j]);

                printf("\n");
            }
                printf("Available resources\n");
                for( j = 0; j < r; j++)
                    printf("%d ", avail[j]);
          safestate();

    }
    }
```

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\n\tEnter
the no of
empty frames: "); scanf("%d",&f);
printf("\n\n\tEnter the length of the string: ");
```

```
scanf("%d",&l); printf("\n\n\tEnter the
string: ");scanf("%s",s);
for(i=0;i<f;i++)F[i]=-
1;

        do
        {
printf("\n\n\t********** MENU **********");
printf("\n\n\t1:FIFO\n\n\t2:LRU \n\n\t4:EXIT");
printf("\n\n\tEnter 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 4:

        exit(0);
        }

printf("\n\n\tDo u want to continue IF YES PRESS 1\n\n\tIF NO PRESS 0 : ");
scanf("%d",&YN);
        }while(YN==1);return(0);
        }
        //FIFO

        void FIFO(char s[],char F[],int l,int f)
        {
        int i,j=0,k,flag=0,cnt=0;
printf("\n\tPAGE\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("\tPage-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("\tNo page-fault");

        }
if(j==f)j=0;
        }
        }
        //LRU
        void lru(char s[],char F[],int l,int f)
        {
int i,j=0,k,m,flag=0,cnt=0,top=0; printf("\n\tPAGE\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("\tPage-fault%d",cnt);cnt++;
        }
        else
printf("\tNo page fault");
flag=0;
        }
        }
```