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.

1a.l

```
%{
#include<stdio.h>
int b=0,op=0,id=0;
%}
%%
[A-Za-z][A-Za-z0-9]*        {id++;printf("\nIdentifier:");ECHO;}
[\+\-\*\/]                  {op++;printf("\nOperator:");ECHO;}
"("                         {b++;}
")"                         {b--;}
.|\n                        {;}
%%
main()
{
        printf("Enter the expression\n");
        yylex();
        printf("\nTotal no. of identifiers%d",id);
        printf("\nTotal no. of operators%d",op);
        if((op+1)==id&&v==0)
                printf("\nExpression is valid\n");
        else
                printf("\nExpression is invalid\n");
}
```

Execution Steps:

```
lex 1a.l
cc lex.yy.c –ll
. /a.out
```

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

1b.l

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

1b.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token NUM
%left '+' '-'
%left '*' '/'
%%
input:exp {printf("%d\n",$1);exit(0);}
exp:exp'+'exp {$$=$1+$3;}
|exp'-'exp {$$=$1-$3;}
|exp'*'exp {$$=$1*$3;}
|exp'/'exp {if($3==0){printf("Divide by zero error\n");exit(0);}
else
$$=$1/$3;}
|'('exp')' {$$=$2;}
```

```
|NUM  {$$=$1;}
;
%%
int main()
{
        printf("Enter the expression\n");
        yyparse();
}
int yyerror()
{
        printf("\nInvalid Expression");
        exit(0);
}
```

Execution Steps:

```
yacc –d 1b.y
lex 1b.l
cc lex.yy.c y.tab.c –ll
. /a.out
```

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 $a^n b$ (note: input *n* value)

2.l

```
%{
#include"y.tab.h"
%}
%%
a       {return A;}
b       {return B;}
\n      ;
.       {return yytext[0];}
%%
```

2.y

```
%{
#include<stdio.h>
int n,count=0;
%}
%token A B
%%
str:str1B {if(count!=n) yyerror();}
|B {if(count!=n) yyerror();}
str1:str1A {count++;}
|A {count++;}
;
%%
main()
{
        printf("Enter the value of n\n");
        scanf("%d",&n);
        printf("Enter the string\n");
        yyparse();
        printf("The string is valid\n");
}
```

```
int yyerror()
{
        printf("Invalid string\n");
        exit(0);
}
```

Execution Steps:

```
yacc –d 2.y
lex 2.l
cc lex.yy.c y.tab.c –ll
. /a.out
```

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

3.c

```
#include<stdio.h>
#include<string.h>
char prod[3][15]={"A->aBa","B->bB","B->@"};
char table[2][3][3]={
{"aBa","",""},
{"@","bB",""}
};                    //Parsing table
int size[2][3]={3,0,0,1,2,0},n;
char s[20],stack[20];
char action[20];
int flag=0;
char c;
void display(int i,int j)
{
        int k;
        if(flag)
        {
        printf("%s",action);
        printf("\n");
        }
        for(k=0;k<=i;k++)
        printf("%c",stack[k]);
        printf("\t");
        for(k=j;k<n;k++)
        printf("%c",s[k]);  // Display contents of input buffer
        printf("\t");
        flag=1;
}
void main()
{
        int i,j,k,row,col;
        printf("\nThe grammar is:\n");
        for(i=0;i<3;i++)
        printf("%s\n",prod[i]);
        printf("\nPredictive parsing table is\n\n");
```

```
printf("\ta\tb\t$\n");
printf("_____\n");       //Display parsing table
for(i=0;i<2;i++)
{
        if(i==0)
                printf("A");
        else
                printf("\nB");
        for(j=0;j<3;j++)
        {
                printf("\t%s",table[i][j]);
        }
}
printf("\nEnter the input string:");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';       // Initializing stack with $
stack[1]='A';       // Push the start symbol A on top of $
i=1;
j=0;
printf("\nStack\tInput\tAction");
printf("\n_____\n");
display(i,j);
while(1)
{
        if(stack[i]==s[j])   // if stack top and current input symbol are same
        {
                strcpy(action,"match");
                c=stack[i];
                strcat(action,&c);
                i--;
                j++;
                if(stack[i]=='$'&&s[j]=='$')     //if both input buffer and stack are empty
                {
                        printf("%s",action);
                        printf("\n$\t$\tSUCCESS\n");
                        break;
```

```
		}
		else
		if(stack[i]=='$'&&s[j]!='$')
		{
			printf("ERROR\n");
			break;
		}
		display(i,j);
}
switch(stack[i])
{
case 'A': row=0;
		break;
case 'B': row=1;
		break;
}
switch(s[j])
{
		case 'a': col=0;
			break;
		case 'b': col=1;
			break;
		case '$': col=2;
			break;
}
if(table[row][col][0]=='\0')
{
		printf("\nERROR\n");
		break;
}
else if(table[row][col][0]=='@')
{
		i--;
		strcpy(action,"OutputB->@");
		display(i,j);
}
else
{
```

```
                        strcpy(action,"Output");
                        if(stack[i]=='A')
                        strcat(action,prod[0]);
                        else
                        strcat(action,prod[1]);
                        for(k=size[row][col]-1;k>=0;k--)
                        {
                                stack[i]=table[row][col][k];
                                i++;
                        }
                        i--;
                        display(i,j);
                }
        }
}
```

Execution Steps:

cc 3.c

./a.out

Output

RUN 1:

The grammar is:

A->aBa

B->bB

B->@

Predictive parsing table is

	a	b	$
A	aBa		
B	@	bB	

Enter the input string: abbba

Stack	Input	Action
$A	abbba$	Output A->aBa
$aBa	abbba$	match a
$aB	bbba$	Output B->bB

$aBb	bbba$	match b
$aB	bba$	Output B->bB
$aBb	bba$	match b
$aB	ba$	Output B->bB
$aBb	ba$	match b
$aB	a$	Output B->@
$a	a$	match a
$	$	SUCCESS

RUN 2:

The grammar is:

A->aBa

B->bB

B->@

Predictive parsing table is

	a	b	$
A	aBa		
B	@	bB	

Enter the input string: ab

Stack	Input	Action
$A	ab$	Output A->aBa
$aBa	ab$	match a
$aB	b$	Output B->bB
$aBb	b$	match b
$aB	$	ERROR

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*.

4.c

```
#include<stdio.h>
#include<string.h>
int z=0,i=0,j=0,c=0;
char a[15],stk[15],act[10];
void check();
void main()
{
        puts("The GRAMMAR is\n E->E+T|T \n T->T*F|F \n F->(E) \n F->id");
        puts("Enter input string:");
        gets(a);
        c=strlen(a);            //Find the length of the string
        strcpy(act,"Shift->");
        puts("\nSTACK \t   INPUT \t\t ACTION");
        printf("_____\n");
        printf("$\t%s$",a);
        for(i=0,j=0;j<c;i++,j++)       //Shift action
        {
                if(a[j]=='i' && a[j+1]=='d')       //For id
                {
                        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\t%sid",stk,a,act);
                        j++;
                        i++;
                        check();
                }
```

```
                else          //For other symbols
                {
                        stk[i]=a[j];
                        stk[i+1]='\0';
                        a[j]=' ';
                        printf("\n$%s\t%s$\t\t%s%c",stk,a,act,stk[i]);
                        check();
                }
        }
        z=0;
        if(stk[z]=='E' &&  strlen(stk)==1)
        {
                printf("\nSUCCESS");
        }
        else
        {
                printf("\nERROR");

        }
}

void check()        //Reduce action
{
        //Reduce by F->(E)
        for(z=0;z<c;z++)
        {
                if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
                {
                        stk[z]='F';
                        stk[z+1]='\0';
                        stk[z+2]='\0';
                        printf("\n$%s\t%s$\t\tReduce (E) to F",stk,a);
                        i=i-2;
```

```
        }
}

//Reduce by F->id
for(z=0;z<c;z++)
{
        if(stk[z]=='i' && stk[z+1]=='d')
        {
                stk[z]='F';
                stk[z+1]='\0';
                printf("\n$%s\t%s$\t\tReduce id to F",stk,a);
                i=i-1;
        }
}

//Reduce by T->T*F|F
for(z=0;z<c;z++)
{
        if(stk[z]=='T' && stk[z+1]=='*' &&  stk[z+2]=='F')
        {
                stk[z]='T';
                stk[z+1]='\0';
                stk[z+2]='\0';
                printf("\n$%s\t%s$\t\tReduce T*F to T",stk,a);
                i=i-2;
          }
        else if(stk[z]=='F')
        {
                stk[z]='T';
                printf("\n$%s\t%s$\t\tReduce F to T",stk,a);
        }
}
```

```
        //Reduce by E->E+T|T
        for(z=0;z<c;z++)
        {
                if(stk[z]=='E' && stk[z+1]=='+' &&  stk[z+2]=='T' && stk[z+3]=='*')
                {
                        break;
                }
                if(stk[z]=='E' && stk[z+1]=='+' &&  stk[z+2]=='T')
                {
                        if(a[j+1]=='*')
                        {
                                break;
                        }
                        else
                        {
                                stk[z]='E';
                                stk[z+1]='\0';
                                stk[z+2]='\0';
                                printf("\n$%s\t%s$\t\tReduce E+T to E",stk,a);
                                i=i-2;
                        }
                }
                else if(stk[z]=='T' && stk[z+1]!='*' && a[j+1]!='*')
                {
                         stk[z]='E';
                        printf("\n$%s\t%s$\t\tReduce T to E",stk,a);
                }
        }
}
```

Execution Steps:

cc 4.c

./a.out

Output:

RUN 1:

GRAMMAR is

E->E+T|T

T->T*F|F

F->(E)

F->ID

Enter input string: id+id*id

STACK	INPUT	ACTION
$	id+id*id$	
$id	+id*id$	Shift->id
$F	+id*id$	Reduce id to F
$T	+id*id$	Reduce F to T
$E	+id*id$	Reduce T to E
$E+	id*id$	Shift->+
$E+id	*id$	Shift->id
$E+F	*id$	Reduce id to F
$E+T	*id$	Reduce F to T
$E+T*	id$	Shift->*
$E+T*id	$	Shift->id
$E+T*F	$	Reduce id to F
$E+T	$	Reduce T*F to T
$E	$	Reduce E+T to E

SUCCESS

RUN 2:

GRAMMAR is

E->E+T|T

T->T*F|F

F->(E)

F->ID

Enter input string: id++

STACK	INPUT	ACTION
$	id++$	
$id	++$	Shift->id
$F	++$	Reduce id to F
$T	++$	Reduce F to T
$E	++$	Reduce T to E
$E+	+$	Shift->+
$E++	$	Shift->+

ERROR

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

5.c

```
#include<stdio.h>
#include<string.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\n",result,arg1,op,arg2); //Read three-address code
		if(strcmp(op,"+")==0)   //Write equivalent machine code to output file
		{
			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,"\nSUB 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,"\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);
}
```

input.txt

T1 –B = ?
T2 C + D
T3 T1 * T2
A T3 = ?

Execution Steps:

cc 5.c
./a.out

Output:

output.txt

```
MOV R0,–B
MOV T1,R0
MOV R0,C
ADD R0,D
MOV T2,R0
MOV R0,T1
MUL R0,T2
MOV T3,R0
MOV R0,T3
MOV A,R0
```

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

6a.l

```
%{
#include<stdio.h>
int ccount=0;
%}
%%
"/*"[^*/]*"*/"          {ccount++;}
"//".*                  {ccount++;}
%%
int main(int argc,char **argv)
{
        FILE *f1,*f2;
        if(argc>1)
        {
                f1=fopen(argv[1],"r");
                if(!f1)
                {
                        printf("Error in opening input file\n");
                        exit(1);
                }
                yyin=f1;
                f2=fopen(argv[2],"w");
                if(!f2)
                {
                        printf("Error in opening output file\n");
                        exit(1);
                }
                yyout=f2;
                yylex();
```

```
                printf("Number of comment lines:%d\n",ccount);
        }
}
```

Input

1.c

```
/* program to add two numbers */
#include<stdio.h>
int main()
{
        int a=10,b=20; //declaring and initializing
        int sum;
        sum=a+b; //computing sum
        printf("sum is %d",sum); //display sum
}
```

Execution Steps:

```
lex 6a.l
cc lex.yy.c  –ll
. /a.out   1.c    2.c
```

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

6b.l

```
%{
#include<stdio.h>
#include"y.tab.h"
%}
%%
[\t]                    ;
[+|-|*|/|=|<|>]      {printf("Operator:%s\n",yytext);return OP;}
[0-9]+               {printf("Numbers:%s\n",yytext);return NUM;}
int|char|bool|float|void|for|do|while|if|else|return    {printf("Keyword:%s\n",yytext);return KEY;}
[a-zA-Z][a-zA-Z0-9]*         {printf("Identifier:%s\n",yytext);return ID;}
\"[^"]*\"                    ;
.                            ;
%%
```

6b.y

```
%{
#include<stdio.h>
#include<stdlib.h>
extern FILE *yyin;
int id=0,dig=0,key=0,op=0;
%}
%token NUM  ID KEY OP
%%
input:NUM  input      {dig++;}
|ID input             {id++;}
|KEY input            {key++;}
|OP input             {op++;}
|NUM                  {dig++;}
```

```
|ID            {id++;}
|KEY           {key++;}
|OP            {op++;}
;
%%
main()
{
        FILE *myfile=fopen("input.c","r");
        if(!myfile)
        {
                printf("Error in opening input.c!");
                return-1;
        }
        yyin=myfile;
        yyparse();
        printf("numbers= %d\nKeywords= %d\nIdentifiers= %d\noperators= %d\n",dig,key,id,
        op);
}
void yyerror()
{
        printf("Parse error!");
        exit(-1);
}
```

Input

input.c

```
#include<stdio.h>
int main()
{
        int a,b;
        printf(“enter a value”);
        scanf(“%d”,&a);
```

```
        printf("enter b value");
        scanf("%d",&b);
        if(a>b)
                printf("a is greater");
        else
                printf("b is greater");
        return 0;
}
```

Execution Steps:

```
lex 6b.l
yacc –d 6b.y
cc lex.yy.c  y.tab.c –ll
. /a.out
```

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.

7.c

```
#include<stdio.h>
#include<stdlib.h>
struct schedule
{
        char pid[20];
        int btime;
        int atime;
        int rtime;
        int stime;
        int etime;
}r[20],temp;
int n,qtime;

void input()
{
        int i;
        printf("PID\t\tAT\t\tBT\n");
        for(i=0;i<n;i++)
        {
                scanf("%s%d%d",r[i].pid,&r[i].atime,&r[i].btime);
                r[i].rtime=r[i].btime;
        }
}
void sort()
{
        int i,j;
        for(i=0;i<n-1;i++)  //Arrange the processes according to remaining time
```

```
		for(j=0;j<n-i-1;j++)
			if(r[j].rtime>r[j+1].rtime)
			{
				temp=r[j];
				r[j]=r[j+1];
				r[j+1]=temp;
			}
		else if(r[j].rtime==r[j+1].rtime)
		{
			if(r[j].atime>r[j+1].atime)
			{
				temp=r[j];
				r[j]=r[j+1];
				r[j+1]=temp;
			}
		}
}

void display()
{
		int i,wt[10],tat[10],total_wt=0,total_tat=0;
		for(i=0;i<n;i++)
		{
			wt[i]=r[i].etime-r[i].btime-r[i].atime;
			total_wt=total_wt+wt[i];
			tat[i]=r[i].btime+wt[i];
			total_tat=total_tat+tat[i];
		}
		printf("\nPID\t\tBT\t\tWT\t\tTT\n");
		for(i=0;i<n;i++)
```

```
	{
		printf("%s\t\t%d\t\t%d\t\t%d\n",r[i].pid,r[i].btime,wt[i],tat[i]);
	}
	printf("Average waiting time is %f\n",(float)total_wt/n);
	printf("Average turnaround time is %f\n",(float)total_tat/n);
}

void srtf()
{
	int t,c=0,i;
	sort();
	t=r[0].atime;
	for(i=1;i<n;i++)		//To find the process that arrived first
		if(t>r[i].atime)
			t=r[i].atime;
	printf("|%d|",t);
	while(c<n)
	{
		i=0;
		//Find a process that has arrived by t and it is not yet finished
		while((r[i].atime>t||r[i].rtime==0)&&i<n)
			i++;
		t++;
		r[i].etime=t;
		r[i].rtime=r[i].rtime-1;
		printf("%s\t\t|%d|",r[i].pid,r[i].etime);
		if(r[i].rtime==0)
			c++;
		sort();
	}
```

```
        display();
}

void rr()
{
        int j=0,c=0,i;
        for(i=0;i<n-1;i++)
                for(j=0;j<n-1-i;j++)
                        if(r[i].atime>r[i+1].atime)
                        {
                                temp=r[i];
                                r[i]=r[i+1];
                                r[i+1]=temp;
                        }
        r[0].stime=r[0].etime=r[0].atime;
        printf("|%d|",r[0].stime);
        while(c<n)
        {
                i=0;
                while(i<n)
                {
                        if(r[i].rtime>qtime)  //if remaining time is greater than quantum time
                        {
                                r[i].stime=r[j].etime;
                                r[i].etime=r[i].stime+qtime;
                                r[i].rtime=r[i].rtime-qtime;
                                printf("%s\t\t|%d|",r[i].pid,r[i].etime);
                                j=i;
                                i++;
                        }
```

```
                else if(r[i].rtime!=0)
                {
                        r[i].stime=r[j].etime;
                        r[i].etime=r[i].stime+r[i].rtime;
                        r[i].rtime=0;
                        printf("%s\t\t|%d|",r[i].pid,r[i].etime);
                        j=i;
                        i++;
                        c++;
                }
                else
                        i++;
        }
    }
    display();
}

int main()
{
    int i,ch;
    printf("Enter the number of processes\n");
    scanf("%d",&n);
    printf("1.SRTF 2.ROUNDROBIN\n");
    printf("Enter your choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1: input();
                srtf();
                break;
```

```
                case 2: input();
                        printf("Enter the quantum time\n");
                        scanf("%d",&qtime);
                        rr();
                        break;
                default: printf("Enter your choice(1or2)");
        }
        return 0;
}
```

Execution Steps:

cc 7.c

./a.out

Output:

RUN1:

```
Enter the number of processes
3
1.SRTF   2.ROUNDROBIN
Enter your choice
1
PID     AT      BT
P1      0       6
P2      2       4
P3      4       2
|0| P1  |1| P1  |2|P1   |3|P1   |4|P1   |5|P1   |6|P3   |7|P3   |8|P2   |9|P2   |10|P2  |11|P2  |12|

PID     BT      WT      TAT
P1      6       0       6
P2      4       6       10
P3      2       2       4
Average waiting time is 2.666667
Average turnaround time is  6.666667
```

RUN2:

Enter the number of processes

3

1.SRTF 2.ROUNDROBIN

Enter your choice

2

PID	AT	BT
P1	0	6
P2	2	4
P3	4	2

Enter the quantum time 2

|0| P1 |2| P2 |4|P3 |6|P1 |8|P2 |10|P1 |12|

PID	BT	WT	TAT
P1	6	6	12
P2	4	4	8
P3	2	0	2

Average waiting time is 3.3333

Average turnaround time is 7.33333

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 p,r,i,j,max[10][10],alloc[10][10],avail[10],need[10][10];
int request[10][10],finish[10],work[10];
int safety_algo()
{
        int k,count=0,flag;
        for(i=0;i<p;i++)
        finish[i]=0;
        for(j=0;j<r;j++)
        work[j]=avail[j];
        while(count<p)
        {
                k=0;
                for(i=0;i<p;i++)
                {
                        flag=0;
                        for(j=0;j<r;j++)
                        if(finish[i]!=0||need[i][j]>work[j])
                        {
                                flag=1;
                                break;
                        }
                        if(flag!=1)
                        {
                                finish[i]=1;
                                for(j=0;j<r;j++)
                                work[j]=work[j]+alloc[i][j];
                                printf("p%d\t",i);
                                count++;
                                k++;
                        }
                }
```

```
                if(k==0)
                {
                        return 0;
                }
        }
        return 1;
}

void resource_request(int i)
{
        int j;
        for(j=0;j<r;j++)
        {
                if(request[i][j]>need[i][j])
                {
                        printf("Error::request more than demand\n");
                        return;
                }
        }
        for(j=0;j<r;j++)
        {
                if(request[i][j]>avail[j])
                {
                        printf("request more than available,process has to wait\n");
                        return;
                }
        }
        for(j=0;j<r;j++)
        {
                alloc[i][j]=alloc[i][j]+request[i][j];
                avail[j]=avail[j]-request[i][j];
                need[i][j]=need[i][j]-request[i][j];
        }
        if(safety_algo()==1)
        printf("\n SAFE::process %d can be allocated\n",i);
```

```
        else
        printf("UNSAFE::process shas towait\n");
}

int main()
{
        printf("enter the number of processes:");
        scanf("%d",&p);
        printf("\n enter the number of resources:");
        scanf("%d",&r);
        printf("\n enter the max matrix\n");
        for(i=0;i<p;i++)
        for(j=0;j<r;j++)
                scanf("%d",&max[i][j]);
        printf("\n enter the alloc matrix\n");
        for(i=0;i<p;i++)
        for(j=0;j<r;j++)
                scanf("%d",&alloc[i][j]);
        printf("\n enter the available resources\n");
        for(j=0;j<r;j++)
                scanf("%d",&avail[j]);
        for(i=0;i<p;i++)
        for(j=0;j<r;j++)
                need[i][j]=max[i][j]-alloc[i][j];
        printf("\tNEED\t\n");
        for(i=0;i<p;i++)
        {
                for(j=0;j<r;j++)
                        printf("%d\t",need[i][j]);
                printf("\n");
        }
        if(safety_algo()==1)
                printf("\n current state of system::SAFE\n");
        else
                printf("\n current state of system::UNSAFE\n");
```

```
        printf("enter the new request(processid)\n");
        scanf("%d",&i);
        printf("\n enter the request\n");
        for(j=0;j<r;j++)
                scanf("%d",&request[i][j]);
        resource_request(i);
        return 0;
}
```

Compilation step:

```
cc      8.c
./a.out
```

RUN1:

```
enter the number of processes:5
enter the number of resources:3
enter the max matrix
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3
enter the alloc matrix
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
enter the available resources
3       3       2
NEED
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
P1      P3      P4      P0      P2
```

current state of system::SAFE

enter the new request(processid)

1

enter the request

1 0 2

SAFE::process 1 can be allocated

RUN2:

enter the number of processes:5

enter the number of resources:3

enter the max matrix

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

enter the alloc matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

enter the available resources

3 3 2

NEED

7 4 3

1 2 2

6 0 0

0 1 1

4 3 1

P1 P3 P4 P0 P2

current state of system::SAFE

enter the new request(processid)

2

enter the request

1 0 2

Error::request more than demand

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>
int n,f;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;
void getData()
{
	printf("\n Enter length of page reference sequence:");
	scanf("%d",&n);
	printf("\n Enter the page reference sequence:\n");
	for(i=0;i<n;i++)
	scanf("%d",&in[i]);
	printf("\nEnter no.of frames:");
	scanf("%d",&f);
}
void initialize()
{
	pgfaultcnt=0;
	for(i=0;i<f;i++)
	p[i]=9999;
}
int isHit(int data)
{
	hit=0;
	for(j=0;j<f;j++)
	{
		if(p[j]==data)
		{
			hit=1;
			break;
		}
	}
return hit;
}

void dispPages()
{
	for(k=0;k<f;k++)
	{
		if(p[k]!=9999)
			printf("%d ",p[k]);
	}
}

void dispPgFaultCnt()
{
	printf("\nTotal no. of pagefaults:%d",pgfaultcnt);
}
```

```
void fifo()
{
	int j=0;
	initialize();
	for(i=0;i<n;i++)
	{
		printf("\nPage%d:",in[i]);
		if(isHit(in[i])==0)
		{
			p[j++]=in[i];
			pgfaultcnt++;
			dispPages();
			if(j==f)
			j=0;
		}
		else
			printf("No. pagefault");
	}
	dispPgFaultCnt();
}

void lru()
{
	initialize();
	int least[50];
	for(i=0;i<n;i++)
	{
		printf("\nPage%d:",in[i]);
		if(isHit(in[i])==0)
		{
			for(j=0;j<f;j++)
			{
				int pg=p[j];
				int found=0;
				for(k=i-1;k>=0;k--)
				{
					if(pg==in[k])
					{
						least[j]=k;
						found=1;
						break;
					}
					else
						found=0;
				}
				if(!found)
					least[j]=-9999;
			}
			int min=9999;
			int repindex;
			for(j=0;j<f;j++)
```

```
                {
                        if(least[j]<min)
                        {
                                min=least[j];
                                repindex=j;
                        }
                }
                p[repindex]=in[i];
                pgfaultcnt++;
                dispPages();
        }
        else
                printf("No. pagefault!");
    }
        dispPgFaultCnt();
}

int main()
{
    int choice;
    while(1)
    {
        printf("\nPage  Replacement Algorithms\n1.Enter data\n2.FIFO\n3.LRU\n4.Exit\nEnter your
        choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                getData();
                break;
            case 2:
                fifo();
                break;
            case 3:
                lru();
                break;
            default:
                return 0;
                break;
        }
    }
}
```

OUTPUT

RUN 1

```
 [root@localhost pgms]# cc 9.c
[root@localhost pgms]# ./a.out
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:1
```

Enter length of page reference sequence:5
Enter the page reference sequence:
1 2 3 4 5
Enter no.of frames:2
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:2
Page1:1
Page2:1 2
Page3:3 2
Page4:3 4
Page5:5 4
Total no. of pagefaults:5
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:3
Page1:1
Page2:1 2
Page3:3 2
Page4:3 4
Page5:5 4
Total no. of pagefaults:5
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:4

RUN 2

[root@localhost pgms]# ./a.out
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:1
Enter length of page reference sequence:15
Enter the page reference sequence:
0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7
Enter no.of frames:
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:^Z

```
[4]+ Stopped ./a.out
[root@localhost pgms]# ./a.out
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:1
Enter length of page reference sequence:16
Enter the page reference sequence:
0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7
Enter no.of frames:3
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:1
Enter length of page reference sequence:16
Enter the page reference sequence:
0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7
Enter no.of frames:3
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:2
Page0:0
Page1:0 1
Page2:0 1 2
Page3:3 1 2
Page0:3 0 2
Page1:3 0 1
Page2:2 0 1
Page3:2 3 1
Page0:2 3 0
Page1:1 3 0
Page2:1 2 0
Page3:1 2 3
Page4:4 2 3
Page5:4 5 3
Page6:4 5 6
Page7:7 5 6
Total no. of pagefaults:16
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:3
Page0:0
```

Page1:0 1
Page2:0 1 2
Page3:3 1 2
Page0:3 0 2
Page1:3 0 1
Page2:2 0 1
Page3:2 3 1
Page0:2 3 0
Page1:1 3 0
Page2:1 2 0
Page3:1 2 3
Page4:4 2 3
Page5:4 5 3
Page6:4 5 6
Page7:7 5 6
Total no. of pagefaults:16
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:4

RUN 3

[root@localhost pgms]# ./a.out
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:1
Enter length of page reference sequence:16
Enter the page reference sequence:
0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7
Enter no.of frames:3
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:2
Page0:0
Page1:0 1
Page2:0 1 2
Page3:3 1 2
Page0:3 0 2
Page1:3 0 1
Page2:2 0 1
Page3:2 3 1
Page0:2 3 0
Page1:1 3 0
Page2:1 2 0
Page3:1 2 3

Page4:4 2 3
Page5:4 5 3
Page6:4 5 6
Page7:7 5 6
Total no. of pagefaults:16
Page Replacement Algorithms
1.Enter data
2.FIFO
3.LRU
4.Exit
Enter your choice:4