

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.

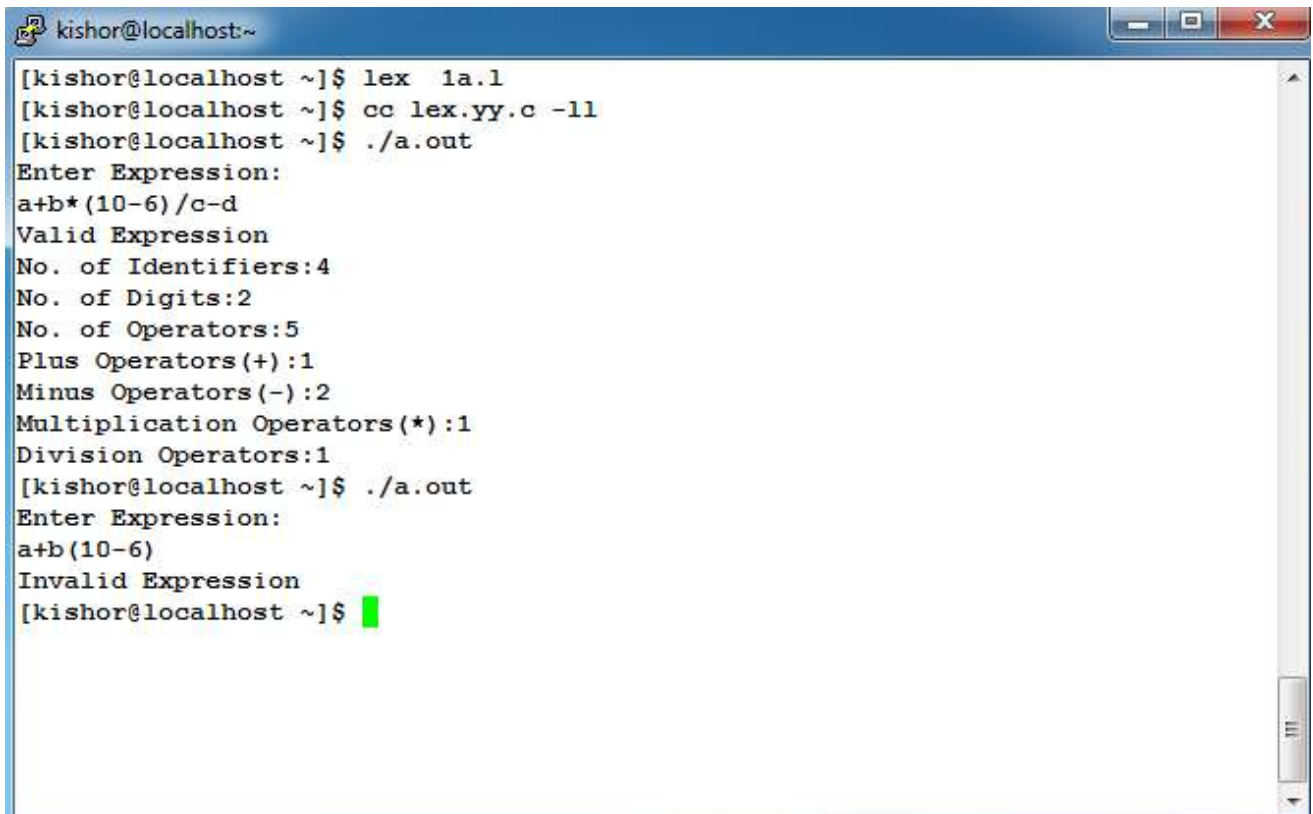
```
%{
    int id=0,d=0,op=0,plus=0,min=0,mul=0,dv=0,valid=1;
}%

%%
[0-9]+ {d++;}
[_a-zA-Z][0-9A-Za-z]* {id++;}
[+] {op++;plus++;}
[-] {op++;min++;}
[*] {op++;mul++;}
[/] {op++;dv++;}
\[ {valid++;}
\] {valid--;}
. ;
[\n] {return 0;}

%%

int main()
{
    printf("Enter Expression:\n");
    yylex();
    if(valid==1&&((d+id)==op+1||op==1))
    {
        printf("Valid Expression\n");
        printf("No. of Identifiers:%d\n",id);
        printf("No. of Digits:%d\n",d);
        printf("No. of Operators:%d\n",op);
        printf("Plus Operators(+):%d\n",plus);
        printf("Minus Operators(-):%d\n",min);
        printf("Multiplication Operators(*):%d\n",mul);
        printf("Division Operators:%d\n",dv);
    }
    else
        printf("Invalid Expression\n");
    return 0;
}
```

OUTPUT:



```
kishor@localhost:~  
[kishor@localhost ~]$ lex 1a.1  
[kishor@localhost ~]$ cc lex.yy.c -ll  
[kishor@localhost ~]$ ./a.out  
Enter Expression:  
a+b*(10-6)/c-d  
Valid Expression  
No. of Identifiers:4  
No. of Digits:2  
No. of Operators:5  
Plus Operators(+):1  
Minus Operators(-):2  
Multiplication Operators(*):1  
Division Operators:/:1  
[kishor@localhost ~]$ ./a.out  
Enter Expression:  
a+b(10-6)  
Invalid Expression  
[kishor@localhost ~]$
```

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

LEX PART :

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

%%

[0-9]+ {yylval=atoi(yytext);return NUM;}
[+]    {return '+';}
[-]    {return '-'}
[*]    {return '*'}
[/]    {return '/'}
[\\t]+ ;
[\\n]   {return 0;}
.       {return yytext[0];}
%%
```

YACC PART :

```
%{
    #include <stdio.h>
    #include <stdlib.h>
}%

%token    NUM
%left     '+' '-'
%left     '*' '/'
```

%%

start: exp {printf("%d\n", \$\$);}

exp: exp+'exp { \$\$ = \$1+\$3; }
 |exp'-'exp { \$\$ = \$1-\$3; }
 |exp'*'exp { \$\$ = \$1*\$3; }
 |exp'/'exp

{

 if(\$3==0)

 {

 yyerror();

 exit(0);

 }

 else

 {

 \$\$ = \$1/\$3;

 }

}

 |>('exp')' { \$\$=\$2; }

 |NUM { \$\$=\$1; }

 ;

%%

int main()

{

 printf("Enter the Expression:\n");

 yyparse();

 return 0;

}

int yyerror()

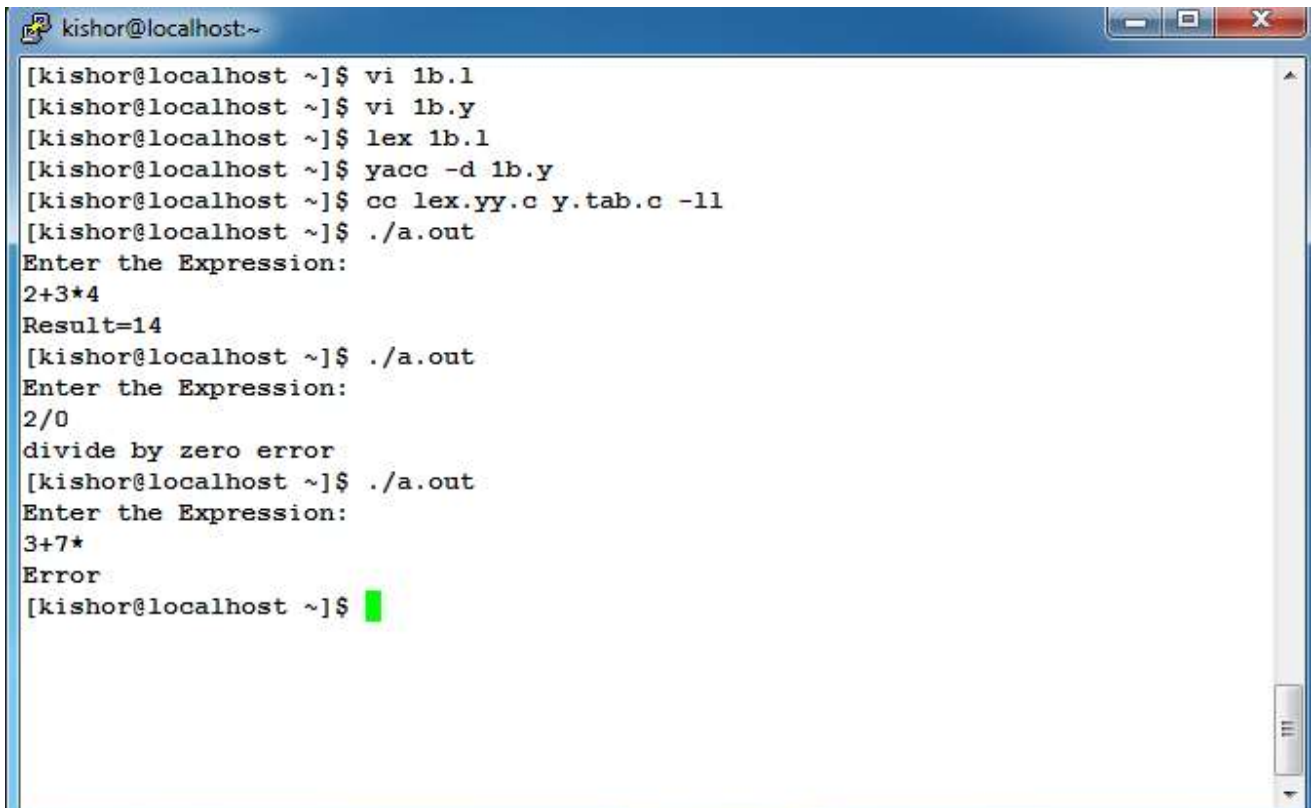
{

 printf("Invalid string\n");

 exit(0);

}

OUTPUT:



```
kishor@localhost:~$ vi 1b.1
[kishor@localhost ~]$ vi 1b.y
[kishor@localhost ~]$ lex 1b.1
[kishor@localhost ~]$ yacc -d 1b.y
[kishor@localhost ~]$ cc lex.yy.c y.tab.c -ll
[kishor@localhost ~]$ ./a.out
Enter the Expression:
2+3*4
Result=14
[kishor@localhost ~]$ ./a.out
Enter the Expression:
2/0
divide by zero error
[kishor@localhost ~]$ ./a.out
Enter the Expression:
3+7*
Error
[kishor@localhost ~]$
```

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 PART :

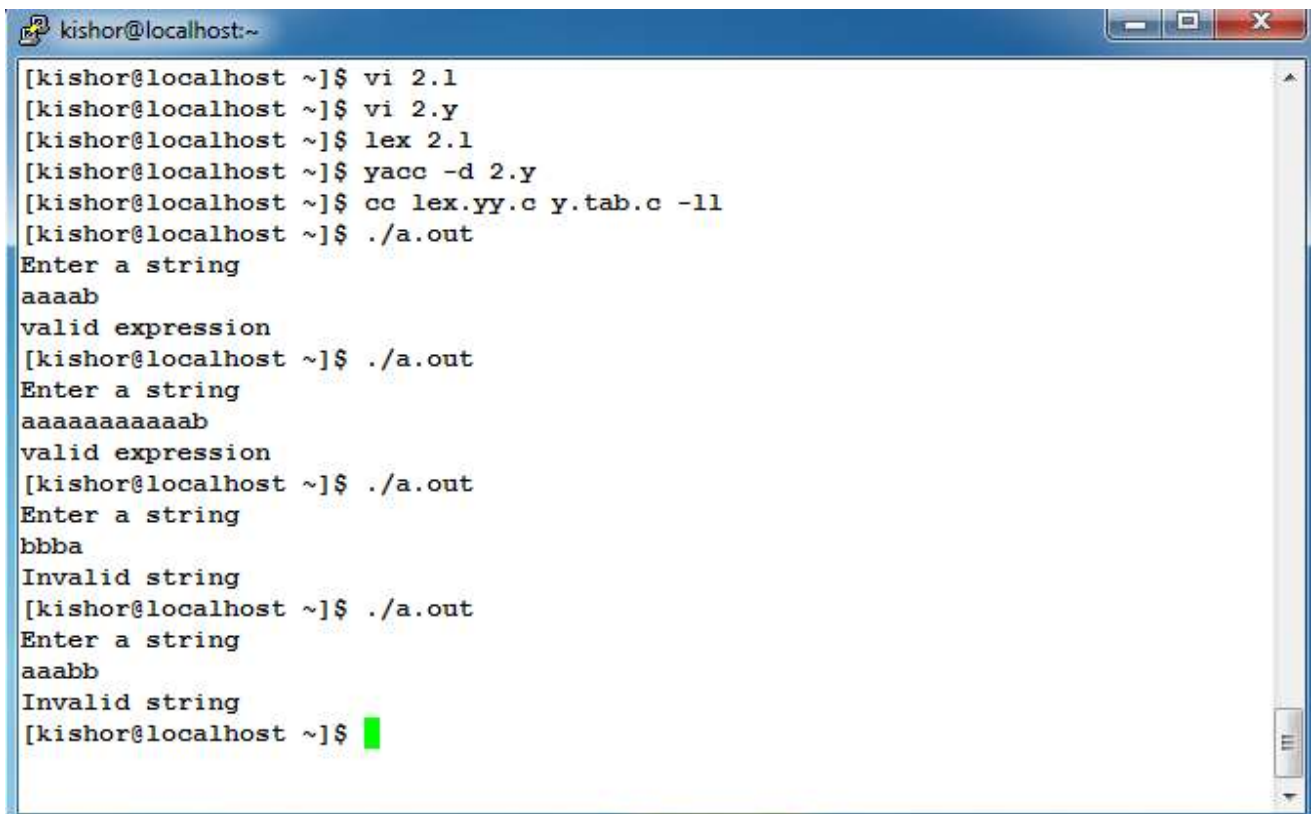
```
%{  
    #include "y.tab.h"  
}%  
  
%%  
  
a      {return A;}  
b      {return B;}  
.      {return yytext[0];}  
[\\n]  {return 0;}  
%%
```

YACC PART :

```
%{  
    #include<stdio.h>  
    #include<stdlib.h>  
}%  
  
%token A B  
  
%%  
  
S      :    A s1 B  
s1     :    A s1  
        |  
        ;  
%%
```

```
int main()
{
    printf("Enter a string \n");
    yyparse();
    printf("valid expression\n");
}
int yyerror()
{
    printf("Invalid string\n");
    exit(0);
}
```

OUTPUT:



```
kishor@localhost:~
[kishor@localhost ~]$ vi 2.1
[kishor@localhost ~]$ vi 2.y
[kishor@localhost ~]$ lex 2.1
[kishor@localhost ~]$ yacc -d 2.y
[kishor@localhost ~]$ cc lex.yy.c y.tab.c -ll
[kishor@localhost ~]$ ./a.out
Enter a string
aaaab
valid expression
[kishor@localhost ~]$ ./a.out
Enter a string
aaaaaaaaaaaaab
valid expression
[kishor@localhost ~]$ ./a.out
Enter a string
bbba
Invalid string
[kishor@localhost ~]$ ./a.out
Enter a string
aaabb
Invalid string
[kishor@localhost ~]$
```

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

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int num(char c)
{
    switch(c)
    {
        case'A':return 0;
        case'B':return 1;
        case'a':return 0;
        case'b':return 1;
        case'@':return 2;
    }
    return 1;
}
int main()
{
    Char
    m[2][3][10]={{"E\0","E\0","E\0"},{"E\0","E\0","E\0"},ip[100],stack[100];
    char
    first[3][10]={"a\0","b\0","@\0"},follow[3][10]={"$\0","a\0","a\0"},LHS[3][3]=
    {"A\0","B\0","B\0"},RHS[3][4]={"aBa\0","bB\0","@\0"};
    int size[2][3]={3,1,1,1,2,1},p,q,r,i,j,n,k,row,col;
    printf("\nfirst={%c,%c,%c}",first[0][0],first[1][0],first[2][0]);
    printf("\nfollow={%c,%c}\n\n",follow[0][0],follow[1][0]);
    for(i=0;i<3;i++)
    {
        if(first[i][0]!='@')
            strcpy(m[num(LHS[i][0])][num(first[i][0])],RHS[i]);
        else
            strcpy(m[num(LHS[i][0])][num(follow[i][0])],RHS[i]);
    }
    printf("Input the String:\n");
    scanf("%s",ip);
    strcat(ip,"$");
```



```
n=strlen(ip);
stack[0]='$';
stack[1]='A';
i=1;j=0;
printf("Parsing Table\n");
for(p=0;p<2;p++)
{
    for(q=0;q<3;q++)
        printf("%s\t",m[p][q]);
    printf("\n");
}
printf("\nStack\tInput\n");
for(k=0;k<=i;k++)
    printf("%c",stack[k]);
printf("\t");
for(k=j;k<=n;k++)
    printf("%c",ip[k]);
printf("\n");
while((stack[i]!='$')&&(ip[j]!='$'))
{
    if(stack[i]==ip[j])
    {
        i--;
        j++;
        for(k=0;k<=i;k++)
            printf("%c",stack[k]);
        printf("\t");
        for(k=j;k<=n;k++)
            printf("%c",ip[k]);
        printf("\n");
    }
    switch(stack[i])
    {
        case 'A': row=0;break;
        case 'B': row=1;break;
        default:
            if((stack[i]=='$')&&(ip[j]=='$'))
                printf("Successful Parsing\n");
    }
}
```

```
        else
            printf("Parsing Error\n");
            exit(0);
    }
    switch(ip[j])
    {
        case 'a': col=0; break;
        case 'b': col=1; break;
        case 'c': col=2; break;
    }
    if(m[row][col][0]==ip[j])
    {
        for(k=size[row][col]-1;k>=0;k--)
        {
            stack[i]=m[row][col][k];
            i++;
        }
        i--;
    }
    if(m[row][col][0]=='E')
    {
        if(i>0)
        {
            printf("Error\n");
            exit(0);
        }
    }
    if(m[row][col][0]=='@')
        i--;
    for(k=0;k<=i;k++)
        printf("%c",stack[k]);
    printf("\t");
    for(k=j;k<=n;k++)
        printf("%c",ip[k]);
    printf("\n");
}
}
```

OUTPUT:

```
kishor@localhost:~
[kishor@localhost ~]$ vi 3.c
[kishor@localhost ~]$ cc 3.c
[kishor@localhost ~]$ ./a.out

first={a,b,@}
follow={$,a}
Input the String:
abba
Parsing Table
aBa      E      E
@        bB      E

Stack    Input
$A       abba$
$aBa     abba$
$aB       bba$
$aBb     bba$
$aB       ba$
$aBb     ba$
$aB       a$
$a        a$
$         $
Successful Parsing
[kishor@localhost ~]$
```

```
[kishor@localhost ~]$ ./a.out

first={a,b,@}
follow={$,a}
Input the String:
AAA
Parsing Table
aBa      E      E
@        bB      E

Stack    Input
$A       AAA$
$        AA$
Parsing Error
[kishor@localhost ~]$
```

4. Design, develop and implement YACC/C program to demonstrate Shift Reduce Parsing technique for the grammar rules: $E \rightarrow E+T \mid T, T \rightarrow T * F \mid F, F \rightarrow (E) \mid id$ 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()
{
    printf("Grammar: \n E->E+T|T \n T->T*F|F \n F->(E)|id\n");
    printf("Enter the input string\n");
    scanf("%s",a);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    printf("Stack \t Input \t\t Action\n");
    printf("$\t %s$",a); //Initial contents of the input buffer
    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%s id",stk,a,act);
            check();
        }
        else
        {
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';
            printf("\n$%s\t%s$\t%s symbol",stk,a,act);
            check();
        }
    }
}
```

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

void check()
{
    strcpy(ac,"REDUCE");
    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%s by F->(E)",stk,a,ac);
            i=i-2;
        }
    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%s by F->id",stk,a,ac);
            j++;
        }
    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%s by T->T*F",stk,a,ac);
            i=i-2;
        }
        else if(stk[z]=='F')
        {
            stk[z]='T';
        }
    }
}
```

```
        printf("\n%s\t%s\t%s by T->F",stk,a,ac);
    }
}
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%s by E->E+T",stk,a,ac);
            i=i-2;
        }
        else if(stk[z]=='T')
        {
            if(stk[z+1]=='*' || a[j+1]=='*')
                break;
            stk[z]='E';
            printf("\n%s\t%s\t%s by E->T",stk,a,ac);
        }
    }
}
```

OUTPUT:

```
kishor@localhost:~$ vi 4.c
[kishor@localhost ~]$ cc 4.c
[kishor@localhost ~]$ ./a.out
Grammar:
  E->E+T|T
  T->T*F|F
  F->(E)|id
Enter the input string
id+id*id
Stack      Input      Action
$          id+id*$
$id        +id*id$    SHIFT-> id
$F         +id*id$    REDUCE by F->id
$T         +id*id$    REDUCE by T->F
$E         +id*id$    REDUCE by E->T
$E+       id*id$     SHIFT-> symbol
$E+id      *id$      SHIFT-> id
$E+F       *id$      REDUCE by F->id
$E+T       *id$      REDUCE by T->F
$E+T*      id$       SHIFT-> symbol
$E+T*id    $         SHIFT-> id
$E+T*F     $         REDUCE by F->id
$E+T       $         REDUCE by T->T*F
$E         $         REDUCE by E->E+T
```

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

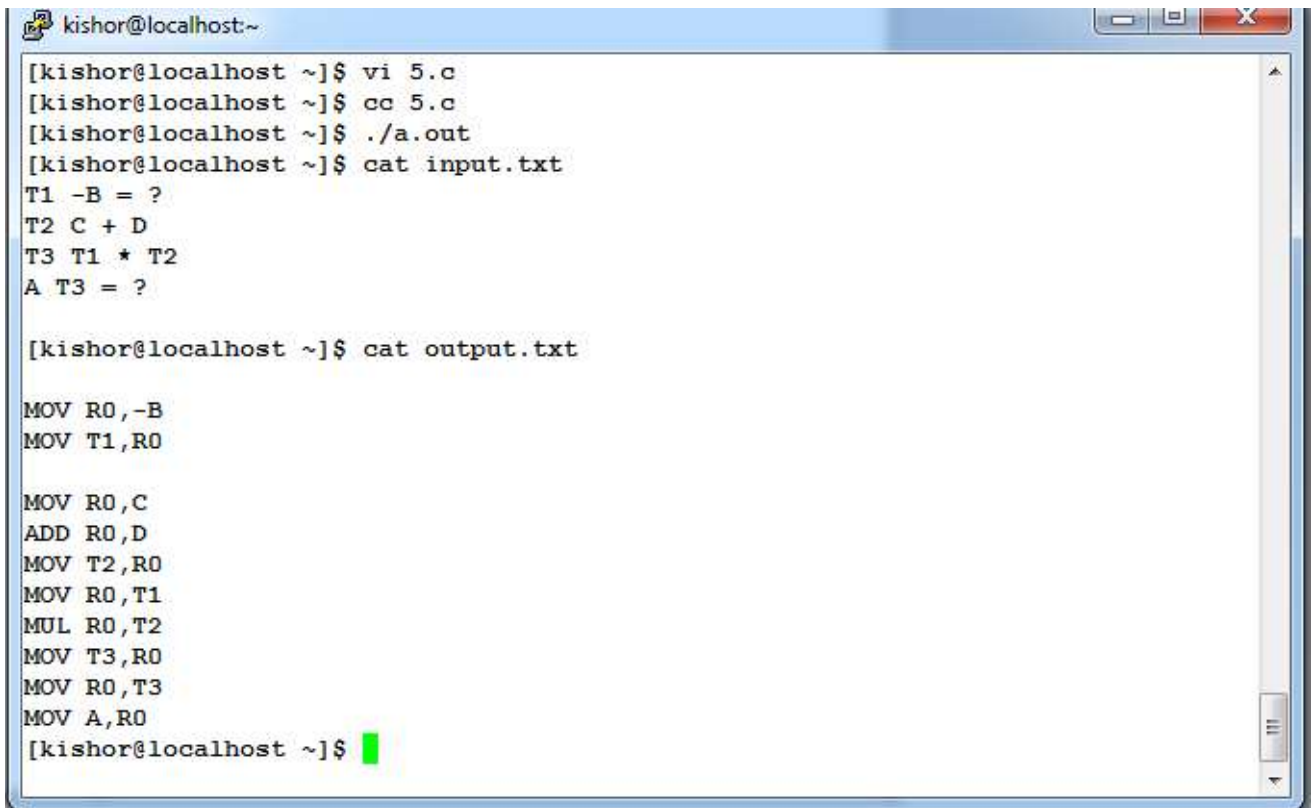
$T1 = -B$
 $T2 = C + D$
 $T3 = T1 + T2$
 $A = T3$

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
int count=0;
int main()
{
    FILE *fp1,*fp2;
    fp1=fopen("input.txt","r");
    fp2=fopen("output.txt","w");
    while(!feof(fp1) && count<4)
    {
        count ++;
        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);
}
```

OUTPUT:

A terminal window titled 'kishor@localhost:~' with standard window controls (minimize, maximize, close) in the top right. The terminal displays a series of commands and their outputs. The user enters 'vi 5.c', 'cc 5.c', and './a.out'. Then, they use 'cat input.txt' to display assembly code. Finally, they use 'cat output.txt' to display the program's output. The assembly code includes instructions for moving values from memory to registers, performing addition and multiplication, and moving the result back to memory. The output shows the results of these operations: -B, C+D, and the product of (-B) and (C+D).

```
kishor@localhost ~]$ vi 5.c
[kishor@localhost ~]$ cc 5.c
[kishor@localhost ~]$ ./a.out
[kishor@localhost ~]$ cat input.txt
T1 -B = ?
T2 C + D
T3 T1 * T2
A T3 = ?

[kishor@localhost ~]$ cat output.txt

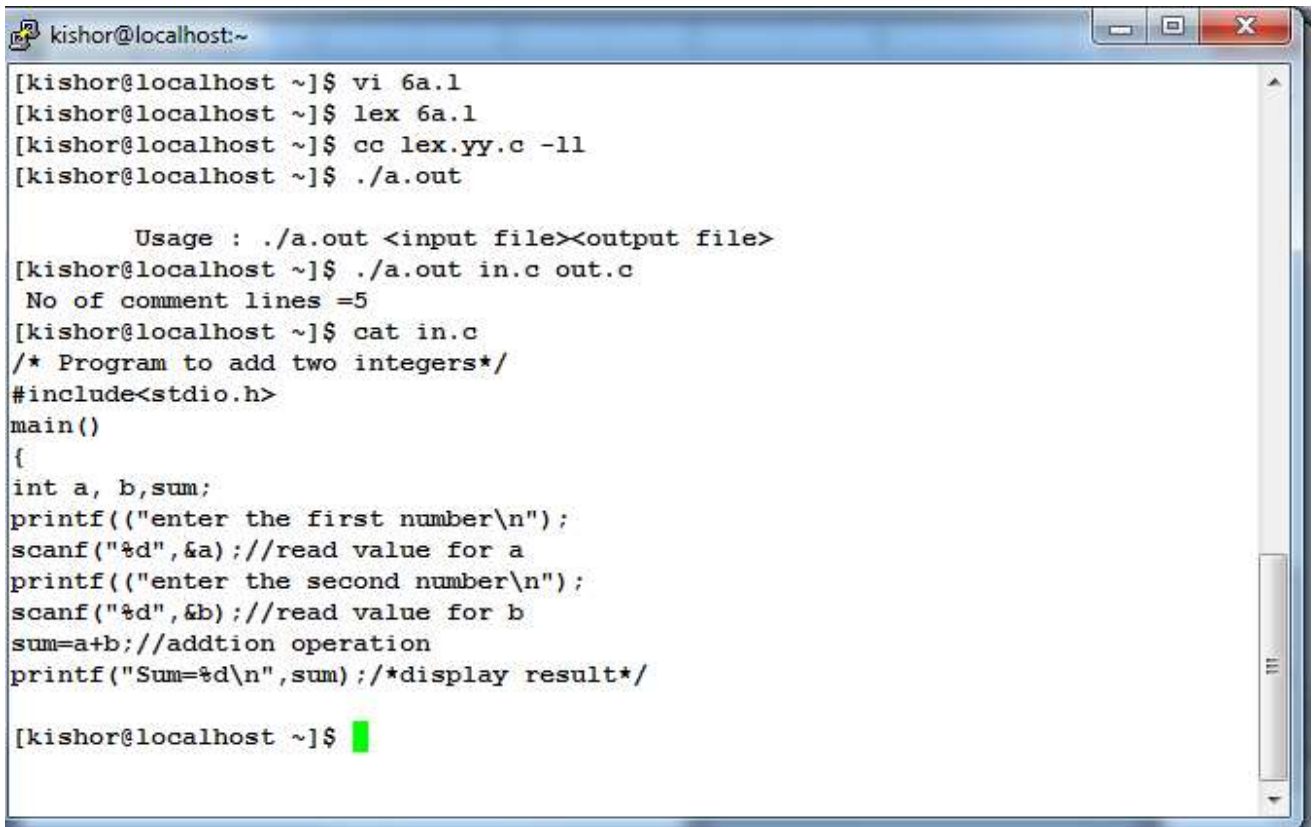
MOV R0,-B
MOV T1,R0

MOV R0,C
ADD R0,D
MOV T2,R0
MOV R0,T1
MUL R0,T2
MOV T3,R0
MOV R0,T3
MOV A,R0
[kishor@localhost ~]$
```

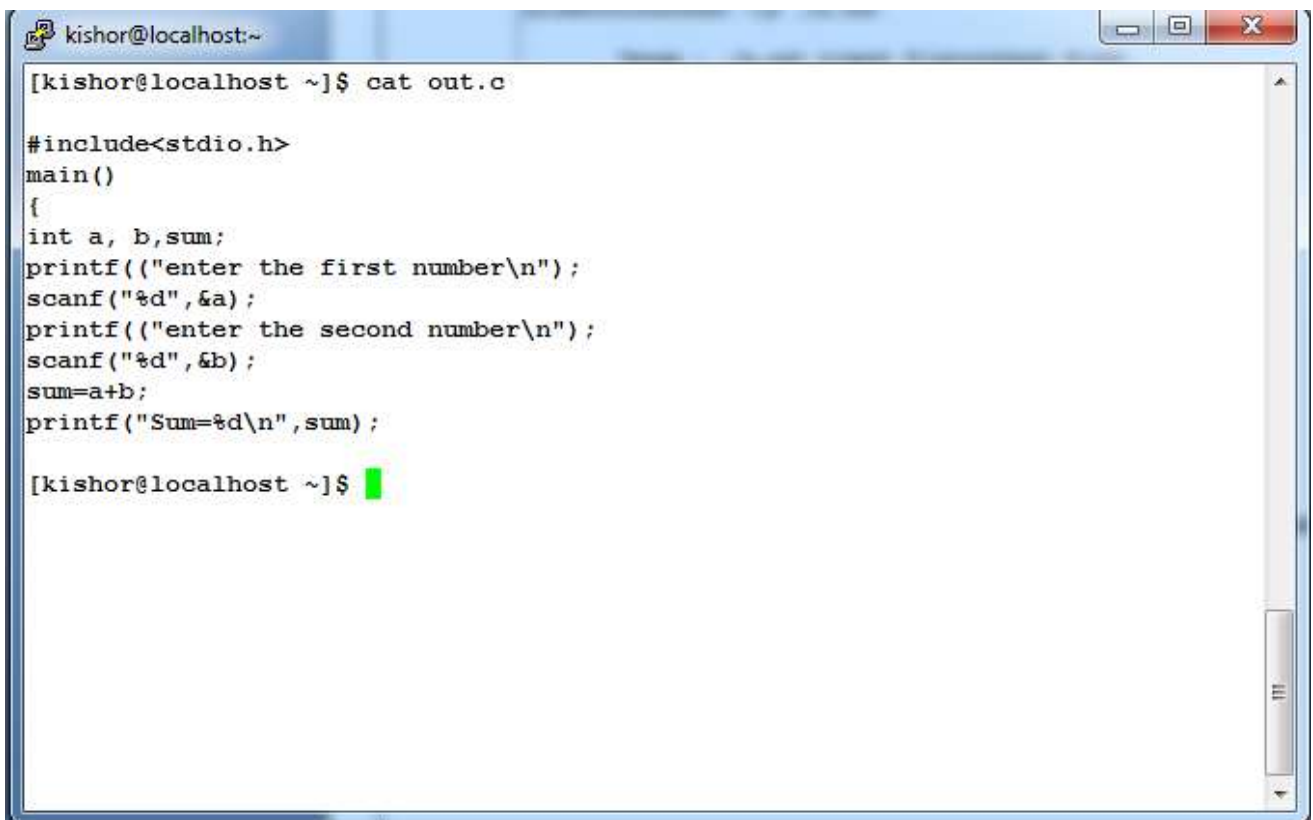
6. a) Write a LEX program to eliminate *comment lines* in a C program and copy the resulting program into a separate file.

```
%{
#include<stdio.h>
int com=0;
}%
%%
"/"[^*]*.*/" {com++;}
\\^[^\\n]*  com++;
%%
main(int argc,char *argv[])
{
if(argc!=3)
{
printf("\n\tUsage : %s <input file><output file>\n", argv[0]);
return;
}
yyin=fopen(argv[1], "r");
yyout=fopen(argv[2], "w");
yylex();
printf(" No of comment lines =%d\n",com);
}
```

OUTPUT:



```
kishor@localhost:~  
[kishor@localhost ~]$ vi 6a.1  
[kishor@localhost ~]$ lex 6a.1  
[kishor@localhost ~]$ cc lex.yy.c -ll  
[kishor@localhost ~]$ ./a.out  
  
Usage : ./a.out <input file><output file>  
[kishor@localhost ~]$ ./a.out in.c out.c  
No of comment lines =5  
[kishor@localhost ~]$ cat in.c  
/* Program to add two integers*/  
#include<stdio.h>  
main()  
{  
int a, b,sum;  
printf(("enter the first number\n");  
scanf("%d",&a);//read value for a  
printf(("enter the second number\n");  
scanf("%d",&b);//read value for b  
sum=a+b;//addition operation  
printf("Sum=%d\n",sum);/*display result*/  
  
[kishor@localhost ~]$
```



```
kishor@localhost:~  
[kishor@localhost ~]$ cat out.c  
  
#include<stdio.h>  
main()  
{  
int a, b,sum;  
printf(("enter the first number\n");  
scanf("%d",&a);  
printf(("enter the second number\n");  
scanf("%d",&b);  
sum=a+b;  
printf("Sum=%d\n",sum);  
  
[kishor@localhost ~]$
```

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 yyval;
}%

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

YACC PART :

```
%{
#include<stdio.h>
#include<stdlib.h>
int id=0,dig=0,key=0,op=0;
}%

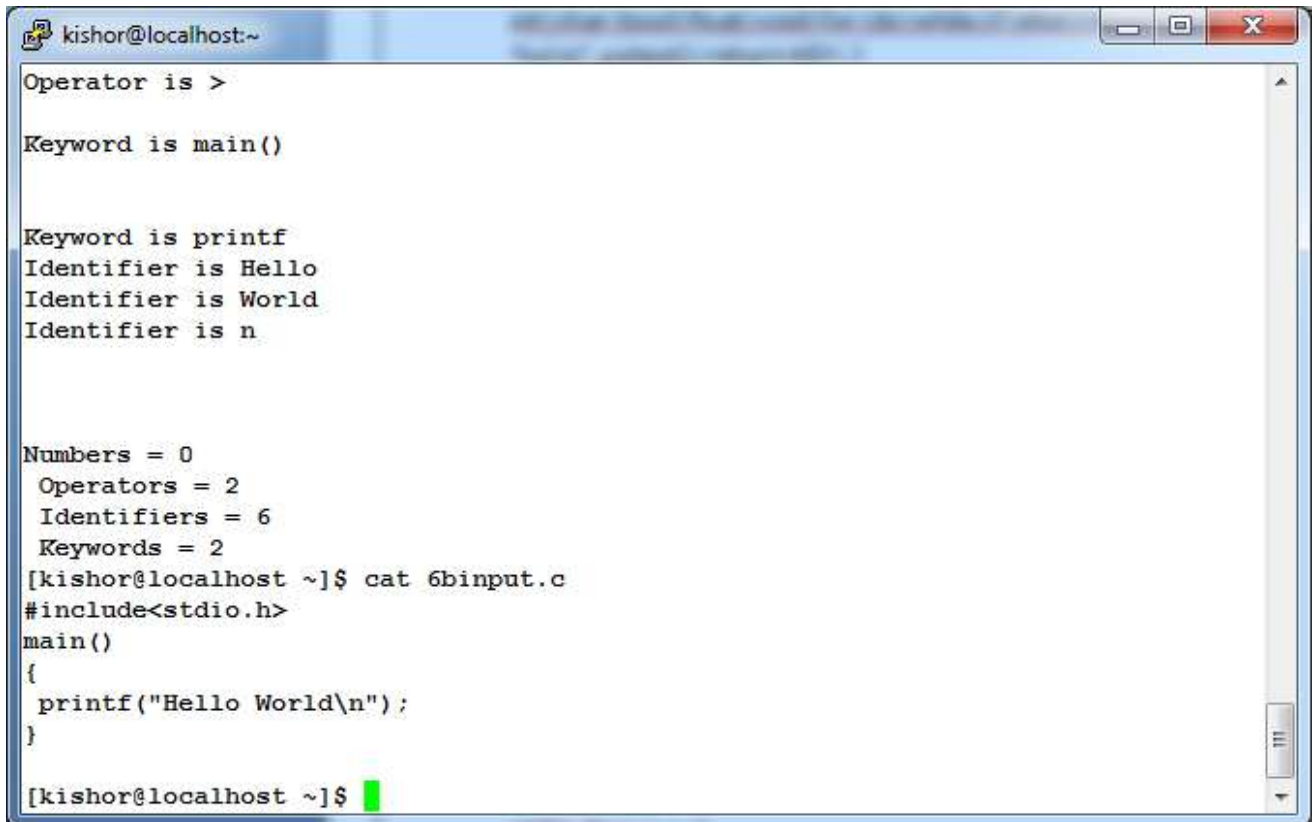
%token DIG ID KEY OP

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

```
|OP {op++;} ;  
%%
```

```
FILE *yyin;  
int main()  
{  
FILE *fp=fopen("6binput.c","r");  
if(!fp)  
{  
printf("File not found\n");  
return -1;  
}  
yyin=fp;  
do{  
    yyparse();  
}while(!feof(yyin));  
printf("Numbers = %d\n Operators = %d\n Identifiers = %d\n Keywords =  
%d\n",dig,op,id,key);  
}  
  
int yyerror()  
{  
printf("Error \n");  
exit(-1);  
}
```

OUTPUT:



A terminal window titled 'kishor@localhost:~' displays the output of a program that counts tokens in a C program. The output lists counts for operators, keywords, identifiers, numbers, and the total count of each. Below the statistics, the content of the file '6binput.c' is shown, which is a simple 'Hello World' program using printf.

```
kishor@localhost:~  
Operator is >  
Keyword is main()  
  
Keyword is printf  
Identifier is Hello  
Identifier is World  
Identifier is n  
  
Numbers = 0  
Operators = 2  
Identifiers = 6  
Keywords = 2  
[kishor@localhost ~]$ cat 6binput.c  
#include<stdio.h>  
main()  
{  
    printf("Hello World\n");  
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
struct J
{
    int id,bt,tat,wt,at,ft;
} job[100];
void scheduler(struct J job[],int n,int q,int c)
{
    int burst[100],x,t=0,done=0,curr=0,diff=q,i=0;
    float tat_sum=0,wt_sum=0;
    for(i=0;i<n;i++)
        burst[i]=job[i].bt;
    if(c==0)
        curr=-1;
    while(done<n)
    {
        if(c==1)
        {
            for(x=0;x<n;x++)
            {
                if(job[curr].bt==0)
                    curr=x;
                if(job[x].bt<job[curr].bt && job[x].bt>0 &&
job[x].at<=t)
                    curr=x;
            }
            diff=1;
        }
        else
        {
            while(1)
            {
                curr=(curr+1)%n;
```



```
        if(job[curr].bt!=0)
            break;
    }
    diff=(q<=job[curr].bt)?q:job[curr].bt;
}
job[curr].bt-=diff;
t+=diff;
if(job[curr].bt==0)
{
    done++;
    job[curr].ft=t;
}
}
if(c==1)
    printf("SRJF Details are \n");
else
    printf("RR Scheduling Details are \n");
for(i=0;i<n;i++)
{
    job[i].bt=burst[i];
    job[i].tat=job[i].ft-job[i].at;
    job[i].wt=job[i].tat-job[i].bt;
    tat_sum+=job[i].tat;
    wt_sum+=job[i].wt;
}
printf("Job\tBT\tAT\tTAT\tWT\n");
for(i=0;i<n;i++)

printf("%d\t%d\t%d\t%d\t%d\n",i+1,job[i].bt,job[i].at,job[i].tat,job[i].wt);
printf("Avg TAT=%f\nAvg WT=%f\n",tat_sum/n,wt_sum/n);
}
void main()
{
    int n,q,c,i;
    printf("Enter the number of processes:\n");
    scanf("%d",&n);
    printf("Enter the arrival time and burst time\n");
    for(i=0;i<n;i++)
```

```
{
    printf("Job%d: ",i+1);
    scanf("%d%d",&job[i].at,&job[i].bt);
}
while(1)
{
    printf("1.RR\n2.SRJF\n3.Exit\n");
    scanf("%d",&c);
    switch(c)
    {
        case 1: printf("Enter time quantum: ");
                scanf("%d",&q);
                scheduler(job,n,q,0);
                break;
        case 2: scheduler(job,n,1,1);
                break;
        case 3: exit(0);
    }
}
}
```

OUTPUT:

```
kishor@localhost:~
[kishor@localhost ~]$ cc 7.c
[kishor@localhost ~]$ ./a.out
Enter the number of processes:
3
Enter the arrival time and burst time
Job1: 0 4
Job2: 0 3
Job3: 0 2
1.RR
2.SRjF
3.Exit
1
Enter time quantum: 1
RR Scheduling Details are
Job      BT      AT      TAT      WT
1         4        0        9        5
2         3        0        8        5
3         2        0        6        4
Avg TAT=7.666667
Avg WT=4.666667
1.RR
2.SRjF
3.Exit
2
```

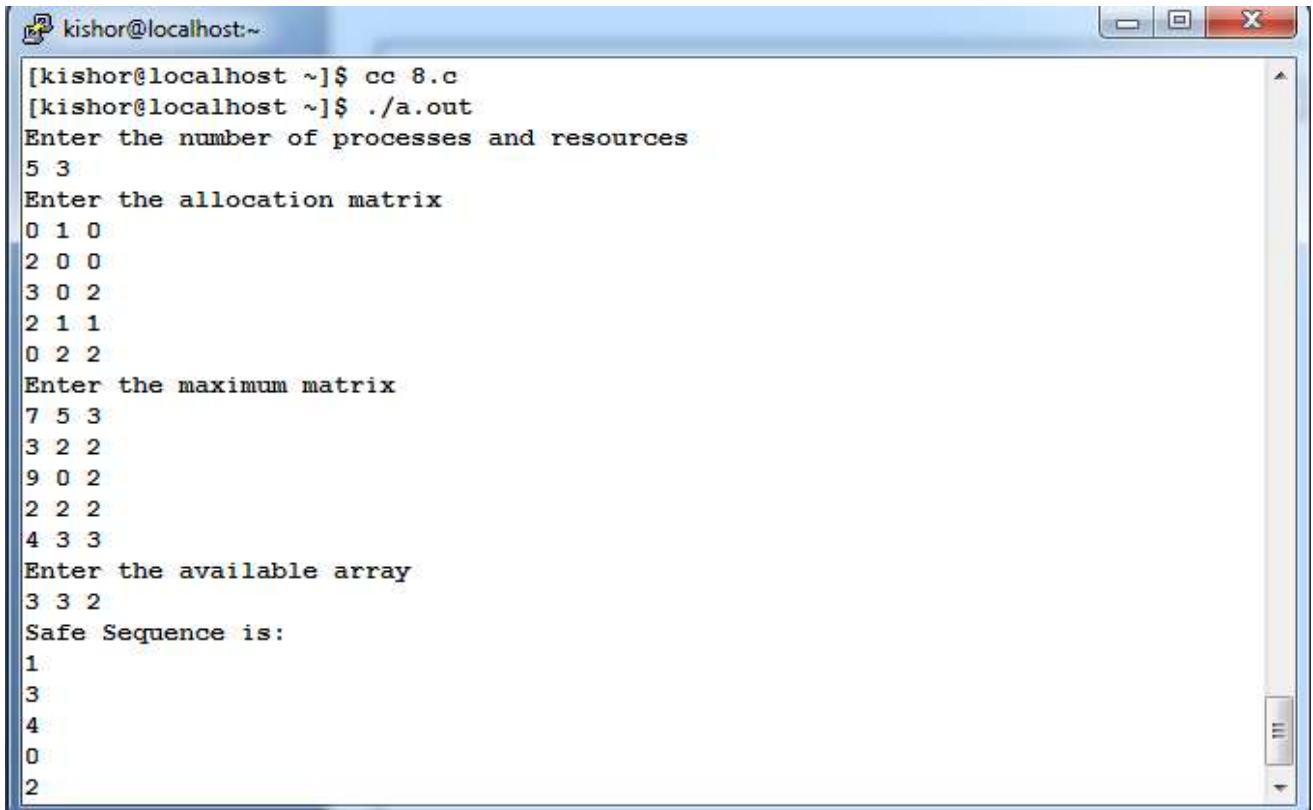
```
SRjF Details are
Job      BT      AT      TAT      WT
1         4        0        9        5
2         3        0        5        2
3         2        0        2        0
Avg TAT=5.333333
Avg WT=2.333333
1.RR
2.SRjF
3.Exit
```

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
alloc[10][10],max[10][10],need[10][10],avail[10],work[10],finish[10],request[
10];
int p,r,j,i,k,v=0,req=0,pno;
int check(int i)
{
    for(j=0;j<r;j++)
        if(need[i][j]>work[j])
            return 0;
    return 1;
}
int main()
{
    printf("Enter the number of processes and resources\n");
    scanf("%d%d",&p,&r);
    int seq[p];
    printf("Enter the allocation matrix\n");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&alloc[i][j]);
    printf("Enter the maximum matrix\n");
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            scanf("%d",&max[i][j]);
    for(i=0;i<p;i++)
        for(j=0;j<r;j++)
            need[i][j]=max[i][j]-alloc[i][j];
    printf("Enter the available array\n");
    for(i=0;i<r;i++)
    {
        scanf("%d",&avail[i]);
```

```
        work[i]=avail[i];
    }
    L1:for(i=0;i<p;i++)
        finish[i]=0;
    while(v<p)
    {
        int allocated=0;
        for(i=0;i<p;i++)
            if(!finish[i]&&check(i))
            {
                for(k=0;k<r;k++)
                    work[k]=work[k]+alloc[i][k];
                allocated=finish[i]=1;
                seq[v]=i;
                v++;
            }
        if(!allocated)
            break;
    }
    for(i=0;i<p;i++)
        if(finish[i]==0)
        {
            printf("Safe Sequence is not generated\n");
            exit(0);
        }
    printf("Safe Sequence is: \n");
    for(i=0;i<v;i++)
        printf("%d\t",seq[i]);
    printf("\n");
}
```

OUTPUT:



```
kishor@localhost:~  
[kishor@localhost ~]$ cc 8.c  
[kishor@localhost ~]$ ./a.out  
Enter the number of processes and resources  
5 3  
Enter the allocation matrix  
0 1 0  
2 0 0  
3 0 2  
2 1 1  
0 2 2  
Enter the maximum matrix  
7 5 3  
3 2 2  
9 0 2  
2 2 2  
4 3 3  
Enter the available array  
3 3 2  
Safe Sequence is:  
1  
3  
4  
0  
2
```

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 s[],char F[],int l,int f)
{
    int i,j=0,k,flag=0;
    printf("PAGE\tFRAMES\tFAULTS");
    for(i=0;i<l;i++)
    {
        for(k=0;k<f;k++)
            if(F[k]==s[i])
                flag=1;
        printf("\n%c\t",s[i]);
        if(flag==0)
        {
            F[j++]=s[i];
            printf("%s",F);
            printf("\tPage Fault");
        }
        else
        {
            flag=0;
            printf("%s",F);
            printf("\tPage Hit");
        }
        if(j==f)
            j=0;
    }
}

void lru(char s[],char F[],int l,int f)
{
    int i,j=0,k,m,flag=0,top=0;
    printf("\nPAGE\t FRAMES\t FAULTS");
    for(i=0;i<l;i++)
    {
        for(k=0;k<f;k++)
```

```
        if(F[k]==s[i])
            flag=1;
        printf("\n%c\t",s[i]);
        if(j!=f && flag!=1)
        {
            F[top]=s[i];
            if(++j!=f)
                top++;
        }
        else
        {
            if(flag!=1)
            {
                for(k=0;k<top;k++)
                    F[k]=F[k+1];
                F[top]=s[i];
            }
            else
            {
                for(m=k;m<top;m++)
                    F[m]=F[m+1];
                F[top]=s[i];
            }
        }
        printf("%s",F);
        if(flag==0)
            printf("\tPage Fault");
        else
            printf("\tPage Hit");
        flag=0;
    }
}

Int main()
{
    int ch,i,l,f;
    char F[10],s[25];
    printf("Enter the no. of frames: ");
    scanf("%d",&f);
```



```
F[f]='\0';
printf("Enter the length of the string: ");
scanf("%d",&l);
printf("Enter the string: ");
scanf("%s",s);
while(1)
{
    printf("\nEnter:\n1:FIFO\n2:LRU\n3:EXIT\n");
    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);
    }
}
```

OUTPUT:

```
kishor@localhost:~
9.c: In function âmainâ:
9.c:69: warning: return type of âmainâ is not âintâ
[kishor@localhost ~]$ ./a.out
Enter the no. of frames: 3
Enter the length of the string: 5
Enter the string: hello

Enter:
1:FIFO
2:LRU
3:EXIT
2

PAGE      FRAMES  FAULTS
h         hÿÿ   Page Fault
e         heÿ   Page Fault
l         hel   Page Fault
l         hel   Page Hit
o         elo   Page Fault

Enter:
1:FIFO
2:LRU
3:EXIT

```

```
kishor@localhost:~
[kishor@localhost ~]$ cc 9.c
9.c: In function âmainâ:
9.c:69: warning: return type of âmainâ is not âintâ
[kishor@localhost ~]$ ./a.out
Enter the no. of frames: 3
Enter the length of the string: 5
Enter the string: hello

Enter:
1:FIFO
2:LRU
3:EXIT
1

PAGE      FRAMES  FAULTS
h         hÿÿ   Page Fault
e         heÿ   Page Fault
l         hel   Page Fault
l         hel   Page Hit
o         oel   Page Fault

Enter:
1:FIFO
2:LRU
3:EXIT
2

```