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.

gedit 1a.l

```
%{
#include<stdio.h>
#include<string.h>
int opc=0,opndc=0,i=0,j=0;
char op[50][20],opnd[50][20];
%}

%%
[-+*/%] {opc++;strcpy(op[i++],yytext);}
[a-zA-Z][a-zA-Z0-9_]* {opndc++;strcpy(opnd[j++],yytext);}
[0-9]+(\.[0-9]+)?(E[-+]?[0-9]+)? {opndc++;strcpy(opnd[j++],yytext);}
\n {return 0;}
%%
main()
{
    int k;
    yylex();
    if((opndc-opc)==1)
            printf("\n Valid Expression");
    else
            printf("\n Invalid Expression");
    printf("\n Number of Operators: %d",opc);
    printf("\n OPERATORS: ");
    for(k=0;k<opc;k++)
            printf("%s ",op[k]);

    printf("\n Number of Operand: %d",opndc);
    printf("\n OPERANDS: ");
    for(k=0;k<opndc;k++)
            printf("%s ",opnd[k]);
    printf("\n");
    return 0;
}
```

Execution:-

lex 1a.l

```
cc lex.yy.c -ll
./a.out

 a+b-c*105
```

Output:-

```
Valid Expression
Number of Operators: 3
OPERATORS: + - *
Number of Operand: 4
OPERANDS: a b c 105
```

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

gedit 1b.y

```
%{
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#define YYSTYPE double
%}
%token num
%left '+' '-'
%left '*' '/'
%right '^'
%right UMINUS
%%
L:E'\n'  {printf("result=%lf\n",$1);exit(0);}
;
E:E'+'E  {$$=$1+$3;}
|E'-'E  {$$=$1-$3;}
|E'*'E  {$$=$1*$3;}
|E'/'E  {$$=$1/$3;}
|E'^'E  {$$=pow($1,$3);}
|'('E')'  {$$=$2;}
|'-' E %prec UMINUS  {$$=-$2;}
|num
;
%%
int main()
{
```

```
    yyparse();
}
yyerror(char *s)
{
    printf("invalid");
    exit(0);
}
yylex()
{
    char c;
    c=getchar();
    if(isdigit(c)|| c=='.')
    {
        ungetc(c,stdin);
        scanf("%lf",&yylval);
        return num;
    }
    return c;

}
```

Execution:-

yacc -d 1b.y
cc y.tab.c -ll -lm
./a.out

Output:-

2*(2^5/8)+10
result=18.000000

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

gedit p2.y

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B
%%
S:T B
|;
T:A T
|;
%%
int main()
{
    yyparse();
    printf("valid\n");
}
yyerror(char *s)
{
    printf("invalid\n");
    exit(0);
}
```

gedit p2.l

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

Execution:-

yacc -d p2.y
lex p2.l
cc y.tab.c lex.yy.c -ll
./a.out

Output:-

aaab
Valid

abb
invalid

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$.

gedit p3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char prod[3][10] = {"A->aBa", "B->bB", "B->@"};
char first[3][4] = {"a", "b", "@"};
char follow[3][4] = {"$", "a", "a"};
char stack[30], input[30], curp[30], table[3][5][20];
int top = -1;
void push(char c)
{
    stack[++top] = c;
}
void pop()
{
    top--;
}
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;
    }
}
void main()
{
    int i, j, k;
    printf("enter string ending with $\n");
    scanf("%s", input);
    for (i = 0; input[i] != '\0'; i++)
        if (input[i] != 'a' && input[i] != 'b' && input[i] != '$')
        {
            printf("invalid string\n");
            exit(0);
        }
    printf("productions \n");
    for (i = 0; i < 3; i++)
        printf("%s\n", prod[i]);
    printf("\n first={%s,%s,%s}\n", first[0], first[1], first[2]);
    printf("\n follow={%s,%s,%s}\n", follow[0], follow[1], follow[2]);
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            strcpy(table[i][j], " ");
    strcpy(table[1][0], "A");
    strcpy(table[2][0], "B");
    strcpy(table[0][1], "a");
    strcpy(table[0][2], "b");
    strcpy(table[0][3], "$");
    strcpy(table[1][1], "A->aBa");
    strcpy(table[2][1], "B->@");
    strcpy(table[2][2], "B->bB");
    printf("table\n");
    for (i = 0; i < 3; i++)
    {
        printf("-----------------------------------------------------");
        for (j = 0; j < 4; j++)
        {
            printf("%s\t", table[i][j]);
        }
        printf("\n");
    }
    push('$');
    push('A');
```

```
printf("stack\tinput\taction\n");
i = 0;
while (input[i] != '$' && stack[top] != '$')
{
    display();
    printf("\t%s", (input + i));
    if (stack[top] == input[i])
    {
        printf("\t matched %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, " ") == 0)
            {
                printf("invalid string\n");
                exit(0);
            }
            else
            {
                printf("\t  apply production %s\n", curp);
                if (curp[3] == '@')
                    pop();
                else
                {
                    pop();
                    k = strlen(curp);
                    for (j = k - 1; j >= 3; j--)
                        push(curp[j]);
                }
            }
        }
        else
        {
            printf("invalide string\n");
            exit(0);
        }
    }
}
display();
```

```
    printf("\t%s\n", (input + i));
    if (input[i] == '$' && stack[top] == '$')
        printf("valid\n");
    else
        printf("invalid\n");
}
```

Execution:-

cc p3.c
./a.out

Output:-

```
enter string ending with $
abba$
productions
A->aBa
B->bB
B->@

 first={a,b,@}

 follow={$,a,a}
table
---------------------------------------------------------------------
        a       b       $
---------------------------------------------------------------------
A       A->aBa
---------------------------------------------------------------------
B       B->@  B->bB
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
```

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

gedit p4.c

```
%{
#include<stdio.h>
#include<stdlib.h>
char stack[30],input[30];
int top=-1,i=0;
%}
%token id
%%
L:E'$'  {
          printf("successful\n");
          exit(0);
        }
;
E:E'+' {
          stack[++top]='+';
          input[i-1]=' ';
          printf("%s\t%s\t Shift +\n",stack,input);
        }
T       {
          stack[top--]=' ';
          stack[top--]=' ';
          printf("%s\t%s\t Reduce E->E+T\n",stack,input);
        }
|T      {
          stack[top]='E';
          printf("%s\t%s\t Reduce E->T\n",stack,input);
        }
;
T:T'*'  {
          stack[++top]='*';
          input[i-1]=' ';
          printf("%s\t%s\t Shift * \n",stack,input);
        }
F       {
          stack[top--]=' ';
          stack[top--]=' ';
          printf("%s\t%s\t Reduce T->T*F\n",stack,input);
        }
|F      {
```

```
         stack[top]='T';
         printf("%s\t%s\t Reduce T->F\n",stack,input);
     }
;
F:'('    {
         stack[++top]='(';
         input[i-1]=' ';
         printf("%s\t%s\t Shift ( \n",stack,input);
     }
E')'     {
         stack[++top]=')';
         input[i-1]=' ';
         printf("%s\t%s\t Shift )\n",stack,input);
         stack[top--]=' ';
         stack[top--]=' ' ;
         stack[top]='F';
         printf("%s\t%s\t Reduce F->(E)\n",stack,input);
     }
|id      {
         stack[top--]=' ';
         stack[top]='F';
         printf("%s\t%s\t Reduce F->id\n",stack,input);
     }
;
%%
main()
{
    printf("enter expression ending with $\n");
    scanf("%s",input);
    printf("stack\tinput\taction\n");
    yyparse();
}
yyerror(char *s)
{
    printf("invalid syntax\n");
}
yylex()
{
    if(input[i]=='i'&&input[i+1]=='d')
    {
        stack[++top]='i';
        stack[++top]='d';
        input[i++]=' ';
        input[i++]=' ';
```

```
        printf("%s\t%s\t Shift id\n",stack,input);
        return id;
    }
    if(input[i] == '+' || input[i] == '*')
        return input[i++];
    if(input[i] == '(' || input[i] == ')')
        return input[i++];
    if(input[i]=='$')
        return input[i++];
}
```

//--execution--

yacc -d p4.y
cc y.tab.c -ll
./a.out

//Input

enter expression ending with $
id+(id*id)$

//Output

stack	input	action
id	*id+id$	Shift id
F	*id+id$	Reduce F->id
T	*id+id$	Reduce T->F
T*	id+id$	Shift *
T*id	+id$	Shift id
T*F	+id$	Reduce F->id
T	+id$	Reduce T->T*F
E	+id$	Reduce E->T
E+	id$	Shift +
E+id	$	Shift id
E+F	$	Reduce F->id
E+T	$	Reduce T->F
E	$	Reduce E->E+T

successful

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>
void main(int argc,char *argv[])
{
    FILE *fp1,*fp2;
    char op[2],ar1[5],ar2[5],res[5];
    fp1 = fopen(argv[1],"r");
    fp2 = fopen(argv[2],"w");
    if(fp1==NULL)
    {
     printf("File not exist");
     exit(0);
    }

    while(1)
    {
     fscanf(fp1,"%s%s%s%s",res,ar1,op,ar2);

     if(strcmp(op,"=")==0)
     {
         fprintf(fp2,"MOV R0,%s\n",ar1);
         fprintf(fp2,"MOV %s,R0\n",res);
     }
     if(strcmp(op,"+")==0)
     {
         fprintf(fp2,"MOV R0,%s\n",ar1);
         fprintf(fp2,"ADD R0,%s\n",ar2);
         fprintf(fp2,"MOV %s,R0\n",res);
     }
     if(strcmp(op,"-")==0)
     {
         fprintf(fp2,"MOV R0,%s\n",ar1);
         fprintf(fp2,"SUB R0,%s\n",ar2);
```

```
            fprintf(fp2,"MOV %s,R0\n",res);
        }
        if(strcmp(op,"*")==0)
        {
            fprintf(fp2,"MOV R0,%s\n",ar1);
            fprintf(fp2,"MUL R0,%s\n",ar2);
            fprintf(fp2,"MOV %s,R0\n",res);
        }
        if(strcmp(op,"/")==0)
        {
            fprintf(fp2,"MOV R0,%s\n",ar1);
            fprintf(fp2,"DIV R0,%s\n",ar2);
            fprintf(fp2,"MOV %s,R0\n",res);
        }
        if(feof(fp1))
            break;
    }
    fclose(fp1);
    fclose(fp2);
}
```

gedit Input.txt

```
t1 -b = #
t2 c + d
t3 t1 * t2
a t3 = #
```

//----Execution---

```
gcc pg5.c
 ./a.out input.txt out.txt
```

output:-

gedit out.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
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.

gedit p6.l

```
%{
#include<stdio.h>
int cc=0;
%}
%%
"//".*\n {cc++;}
"/*"([^*]|"*"+[^*/])*"*/"  { cc++; }
%%
void main(int argc, char *argv[])
{
    yyin=fopen(argv[1],"r");
    yyout=fopen(argv[2],"w");
    yylex();
    printf("count=%d",cc);
}
```

Input:-

gedit cinput.txt

```
//Single line comment
%{
#include<stdio.h>
#include<stdlib.h>
this is normal text
%}

/*
Multiline
comment2
*/

/*
This is multiline--
comment
*/
```

Execution:-

```
lex p6.l
cc lex.yy.c -ll
./a.out cinput.txt coutput.txt
```

Output:-

```
count=3
gedit output.txt
```

```
%{
#include<stdio.h>
#include<stdlib.h>
this is normal text
%}
```

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

gedit p6b.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int kc=0,idc=0,nc=0,opc=0;
extern FILE *yyin;
%}

%token id key num op

%%
S :id S {idc++;}
| num S {nc++;}
| key S {kc++;}
| op S {opc++;}
| id {idc++;}
| num {nc++;}
| key {kc++;}
| op {opc++;}
;
%%
```

```
void  main(int argc,char *argv[])
{
    yyin=fopen(argv[1],"r");
    yyparse();
    printf("key count=%d operator count=%d id count=%d number count=%d
\n",kc,opc,idc,nc);
}
int yyerror(char *S)
{
    printf("invalid");
    exit(0);
}
```

gedit p6b.l

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

%}
%%
int|float|char|void|if|switch {return key;}
[-+*/=] {return op;}
[0-9]+(\.[0-9]+)? {return num;}
[a-zA-Z_][a-zA-Z0-9_]* {return id;}
\n {;}
[,:\t: ] {;}
%%
```

Input:-
gedit p6b.txt

```
int float char
205 abc5 22
+ / int
aa bb
```

Execution:-
lex p6b.l
yacc -d p6b.y
cc y.tab.c lex.yy.c -ll
./a.out p6b.txt
Output:-
Key count=4 operator count =2 id count=3 number count=2

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.

gedit p7.c

```
#include<stdio.h>
#include<stdlib.h>
void roundrobin();
void srtf();
int main()
{
    int ch;
    while(1)
    {
     printf("\n1. RR\n 2. SRTF \n 3. Exit\n");
     printf("Enter choice: ");
     scanf("%d",&ch);
     switch(ch)
     {
         case 1: printf("----------------Round Robin----------\n");
                 roundrobin();
                 break;
         case 2: printf("-----------------SRTF-----------------\n");
                 srtf();
                 break;
         case 3: exit(0);
     }
    }
}

void srtf()
{
    int a[10],b[10],r[10],waiting[10],turnaround[10],completion[10],i,j,
      smallest,count=0,time,n;
    float avg=0,tt=0,end;
    printf("Enter the number of processes: ");
    scanf("%d",&n);
    printf("Enter arrival time: ");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter burst time: ");
    for(i=0;i<n;i++)
        scanf("%d",&b[i]);
    for(i=0;i<n;i++)
```

```
        r[i]=b[i];
    b[9]=9999;
    for(time=0;count!=n;time++)
    {
        smallest=9;
        for(i=0;i<n;i++)
            if(a[i]<=time && b[i]<b[smallest] && b[i]>0)
                smallest=i;
        b[smallest]--;
        if(b[smallest]==0)
        {
            count++;
            end=time+1;
            completion[smallest]=end;
            waiting[smallest]=end-a[smallest]-r[smallest];
            turnaround[smallest]=end-a[smallest];
        }
    }
    printf("Process id \t BT  \t AT  \t WT  \t TAT  \t CT\n");
    for(i=0;i<n;i++)
    {
        printf("\n%d \t %d \t %d \t %d \t %d \t
%d",i+1,r[i],a[i],waiting[i],turnaround[i],completion[i]);
        avg+=waiting[i];
        tt+=turnaround[i];
    }
    printf("\nAvg TAT=%f \n Avg WT=%f\n",tt/n,avg/n);
}

void roundrobin()
{
    int n,tq,bt[10],st[10],time=0,tat[10],wt[10],i,count=0,swt=0,stat=0,temp1,
      sq=0,j,k;
    float awt=0.0,atat=0.0;
    printf("Enter the number of procesess: \n");
    scanf("%d",&n);
    printf("Enter the Burst time: \n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&bt[i]);
        st[i]=bt[i];
    }
    printf("Enter the time quantum: \n");
    scanf("%d",&tq);
```

```
    while(1)
    {
        for(i=0,count=0;i<n;i++)
        {
            temp1=tq;
            if(st[i] == 0)
            {
                count++;
                continue;
            }
            if(st[i] > tq)
                st[i] -=tq;
            else if(st[i]>=0)
            {
                temp1 = st[i];
                st[i]=0;
            }
            sq+=temp1;
            tat[i] = sq;
        }
        if(n==count) break;
    }
    for(i=0;i<n;i++)
    {
        wt[i]=tat[i]-bt[i];
        swt +=wt[i];
        stat += tat[i];
    }
    awt = (float)swt/n;
    atat =(float)stat/n;
    printf("Process No \t Burst Time \t Wait Time \t 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 WT=%f AVG TAT = %f",awt,atat);
}
```

Execution:-

gcc p7.c

./a.out

//==========OUTPUT==========

1. RR
2. SRTF
3. Exit

Enter choice: 1
----------Round Robin----------
Enter the number of procesess:
5
Enter the Burst time:
10 1 2 1 5
Enter the time quantum:
2

Process No	Burst Time	Wait Time	Turn Around Time
1	10	9	19
2	1	2	3
3	2	3	5
4	1	5	6
5	5	10	15

AVG WT=5.800000 AVG TAT = 9.600000
1. RR
2. SRTF
3. Exit
Enter choice: 2
----------SRTF----------
Enter the number of processes: 5
Enter arrival time: 0 0 3 5 10
Enter burst time: 10 1 2 1 5

Process id	BT	AT	WT	TAT	CT
1	10	0	4	14	14
2	1	0	0	1	1
3	2	3	0	2	5
4	1	5	0	1	6
5	5	10	4	9	19

Avg TAT=5.4.000000
Avg WT=8.000000

1. RR
2. SRTF
3. Exit
Enter choice: 3

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

gedit p8.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Max[10][10], need[10][10], alloc[10][10], avail[10],
        completed[10], safeSequence[10];
    int p, r, i, j, process, count;
    count = 0;
    printf("Enter the no of processes : ");
    scanf("%d", &p);
    for (i = 0; i < p; i++)
        completed[i] = 0;
    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];
    do
    {
        printf("\n Max matrix:\tAllocation 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("\n");
    }
    process = -1;
    for (i = 0; i < p; i++)
    {
        if (completed[i] == 0)     // if not completed
        {
            process = i;
            for (j = 0; j < r; j++)
            {
                if (avail[j] < need[i][j])
                {
                    process = -1;
                    break;
                }
            }
        }
        if (process != -1)
            break;
    }

    if (process != -1)
    {
        printf("\nProcess %d runs to completion!", process + 1);
        safeSequence[count] = process + 1;
        count++;
        for (j = 0; j < r; j++)
        {
            avail[j] += alloc[process][j];
            alloc[process][j] = 0;
            Max[process][j] = 0;
            completed[process] = 1;
        }
    }
} while (count != p && process != -1);
//end of do while loop
```

```
    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");
    }
    else
        printf("\nThe system is in an unsafe state!!");
}
```

Execution:-

gcc p8.c

./a.out

//==========OUTPUT==========

```
Enter the no of processes : 5


Enter the no of resources : 3


Enter the Max Matrix for each process :
For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 7 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3


Enter the allocation for each process :
For process 1 : 0 1 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2
```

```
Enter the Available Resources :3 3 2

 Max matrix:    Allocation matrix:
7 5 3           0 1 0
3 2 2           2 0 0
7 0 2           3 0 2
2 2 2           2 1 1
4 3 3           0 0 2

Process 2 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3           0 1 0
0 0 0           0 0 0
7 0 2           3 0 2
2 2 2           2 1 1
4 3 3           0 0 2

Process 3 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3           0 1 0
0 0 0           0 0 0
0 0 0           0 0 0
2 2 2           2 1 1
4 3 3           0 0 2

Process 4 runs to completion!
 Max matrix:    Allocation matrix:
7 5 3           0 1 0
0 0 0           0 0 0
0 0 0           0 0 0
0 0 0           0 0 0
4 3 3           0 0 2

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0 0 0           0 0 0
0 0 0           0 0 0
0 0 0           0 0 0
0 0 0           0 0 0
4 3 3           0 0 2

Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence : < 2 3 4 1 5 >
```

9.c

```c
#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("\nEnter the no of empty frames: ");
    scanf("%d",&f);
    printf("\nEnter the length of the string: ");
    scanf("%d",&l);
    printf("\nEnter the string: ");
    scanf("%s",s);
    for(i=0;i<f;i++)
        F[i]='\0';

    while(1)
    {
        printf("\n***** MENU *****");
        printf("\n1:FIFO\n2:LRU \n3:EXIT");
        printf("\nEnter your choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: for(i=0;i<f;i++)
                        F[i]='\0';
                    FIFO(s,F,l,f);
                    break;
            case 2: for(i=0;i<f;i++)
                        F[i]='\0';
                    lru(s,F,l,f);
                    break;
            case 3: exit(0);
            default: printf("invalid");
        }
    }
    return(0);
}
```

```
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\t\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;
    }
}

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\t\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;
    }
}
```

Execution:-
gcc p8.c
./a.out

//==========OUTPUT==========

Enter the no of empty frames: 3

Enter the length of the string: 6

Enter the string: habibi

***** MENU *****
1:FIFO
2:LRU
3:EXIT
Enter your choice: 1

PAGE	FRAMES	FAULTS
h	h	Page-fault 0
a	h a	Page-fault 1
b	h a b	Page-fault 2
i	i a b	Page-fault 3
b	i a b	No page-fault
i	i a b	No page-fault

***** MENU *****
1:FIFO
2:LRU
3:EXIT
Enter your choice: 2

PAGE	FRAMES	FAULTS
h	h	Page-fault 0
a	h a	Page-fault 1
b	h a b	Page-fault 2
i	a b i	Page-fault 3
b	a i b	No page fault
i	a b i	No page fault

***** MENU *****
1:FIFO
2:LRU
3:EXIT
Enter your choice: 3