Program-1

Question: 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.

Program:

[lab1a.l]

```
%{
#include<stdio.h>
int v=0,op=0,id=0,flag=0;
%}

%%
[a-zA-Z]+[0-9A-Za-z]* {id++;}
[0-9]+ {id++;}
[\+\-\*/\=] {op++;}
"(" {v++;}
")" {v--;}
";" {flag=1;}
.|\n {return 0;}
%%

int main()
{
        printf("Enter The Expression:");
        yylex();
        if((op+1)==id && v==0 && flag==0)
        {
                printf("\n Expression Is Valid\n");
                printf("Number Of Identifier = %d \n
                     Number Of Operators = %d
                     \n",id,op);
        }
        else
                printf("\n Expression Is
                     Invalid\n");
```

```
return 0;

}
```

Output:

```
Enter The Expression:1+6*(9-2+4)

 Expression Is Valid
Number Of Identifiers = 4
Number Of Operators = 5

Enter The Expression:3-6+90

 Expression Is Valid
Number Of Identifiers = 3
Number Of Operators = 2
```

Program-1

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

Program:

[lab1b.y]

```
%
{
#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;}
%%

%{
#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 Error\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:

```
Enter An Expression:
6+9
15

Enter An Expression:
25-10
15

Enter An Expression:
3*5
15

Enter An Expression:
(2+3)-(1*8)
-2
```

Program-2

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

Program:

[lab2.l]

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

[lab2.y]

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token A B
%%
input:s'\n' {printf("Successful
                    Grammar\n");exit(0);}
s: A s1 B| B s1: ; | A s1
%%
main()
{
 printf("Enter A String\n"); yyparse();
}

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

Output:

```
Enter A String
ababa
Error

Enter A String
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab
Succesful Grammar

Enter A String
abaaaaaaab
Error

Enter A String
b
Succesful Grammar
```

Program-3

Question: 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$

Program:

[lab3.c]

```
/*GRAMMER RULES ---- A ->aBa , B ->bB | @*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

char prod [3][10]={"A->aBa","B->bB","B->@"};
char first[3][10]={"a","b","@"};
char follow[3][10]={"$","a","a"};
char table[3][4][10];

char input[10];
int top=-1;
char stack[25];
char cmp[20];

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

```
}

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

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

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

                        printf("\nfirst={%s,%s,%s}",fir
                        st[0],first[1],first[2]);
  printf("\nfollow={%s,%s}\n",follow[0],follow[1]);
```

```
printf("\nPredictive Parsing Table For The Given
                    Grammar :\n");

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

for(i=0;i<3;i++)
{
  if(first[i][0]!='@')

                    strcpy(table[numr(prod[i][0])][
                    numr(first[i][0])],prod[i]);
  else

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

```
 printf("Enter The Input String Terminated With $
                        To Parse:-");
 scanf("%s",input);
 for(i=0;input[i]!='\0';i++){

                        if((input[i]!='a')&&(input[i]!=
                        'b')&&(input[i]!='$'))
   {
     printf("Invalid String");
     exit(0);
   }
 }

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

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

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

while(input[i]!='$'&&stack[top]!='$')
 {
   display();
   printf("\t\t%s\t",(input+i));
```

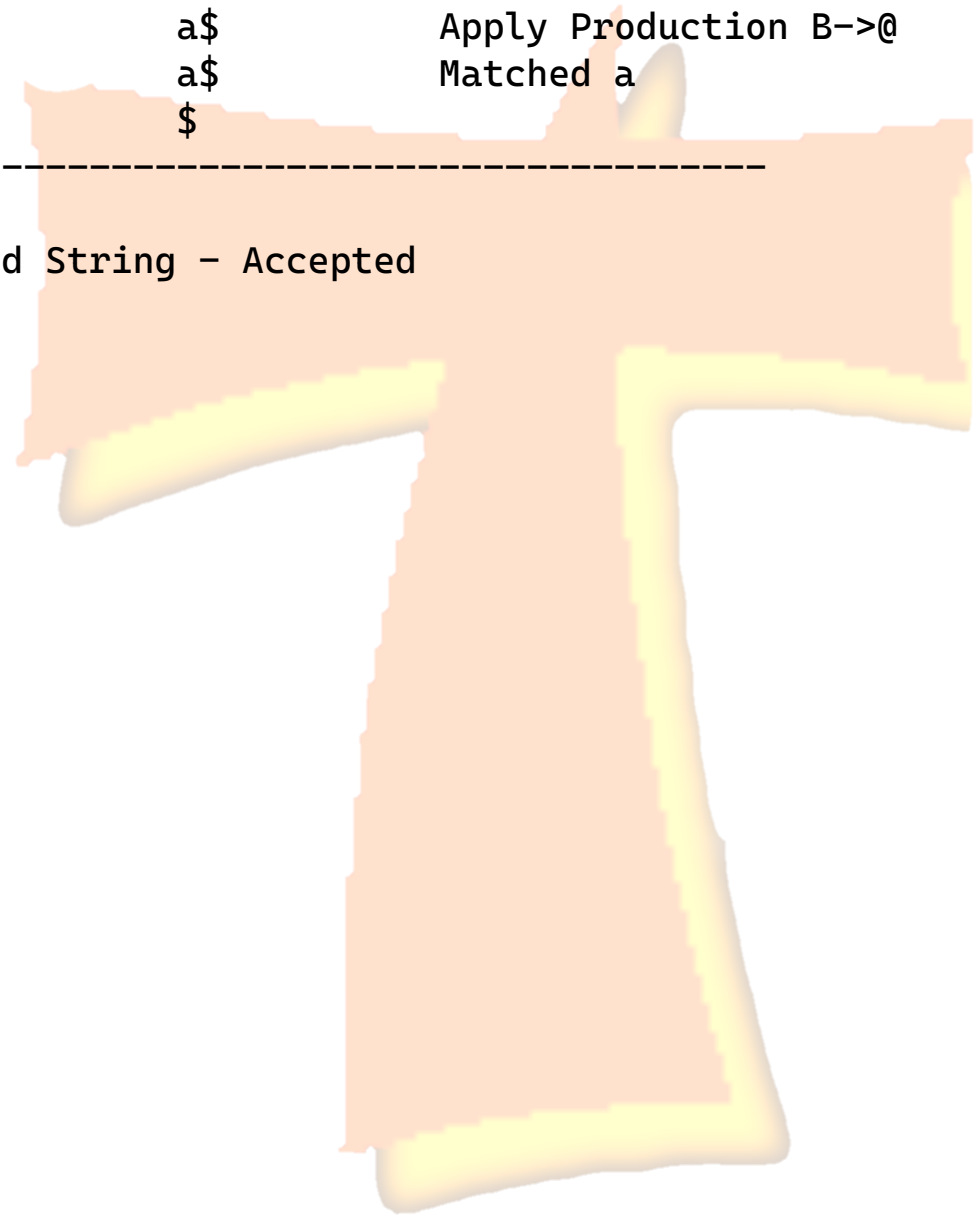
```
    if(stack[top]==input[i])
    {
      printf("\tMatched %c\n", input[i]);
      pop();
      i++;
    }
    else
    {
      if(stack[top]>=65&&stack[top]<92)
      {

                        strcpy(curp,table[numr(stack[to
                        p])][numr(input[i])]);
        if(!(strcmp(curp,"e")))
        {
          printf("\nInvalid String - Rejected\n");
          exit(0);
        }
        else
        {
          printf("\tApply Production %s\n",curp);
          if(curp[3]=='@')
          pop();
          else
          {
            pop();
            n=strlen(curp);
            for(j=n-1;j>=3;j--)
            push(curp[j]);
          }
        }
      }
    }
}
```

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

Output:

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

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

Predictive Parsing Table For The Given Grammar:

---------------------------------------
                a               b
---------------------------------------
A               A->aBa          EMPTY
---------------------------------------
B               B->@            B->bB
---------------------------------------
Enter The Input String Terminated With $ To Parse:-
abba$

Stack       Input       Action
```

```
---------------------------------------
A$          abba$       Apply Production A->aBa
aBa$        abba$       Matched a
Ba$         bba$        Apply Production B->bB
bBa$        bba$        Matched b
Ba$         ba$         Apply Production B->bB
bBa$        ba$         Matched b
Ba$         a$          Apply Production B->@
a$          a$          Matched a
$           $
---------------------------------------

Valid String - Accepted
```



Program-4

Question: 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.

Program:

[lab4.c]

```
#include<stdio.h>
#include<string.h>

int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();

int main()
{
    puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n
                    E->id");
    puts("\nEnter Input String :");
    gets(a);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    puts("Stack \t Input \t Action");
    for(k=0,i=0; j<c; k++,i++,j++)
    {
        if(a[j]=='i' && a[j+1]=='d')
        {
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
            printf("\n$%s\t%s$\t%sid",stk,a,act);
            check();
        }
        else
```

```
        {
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';

                    printf("\n$%s\t%s$\t%ssymbols",
                    stk,a,act);check();
        }
    }
}

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

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

}
```

Output:

```
Grammar is E->E+E
 E->E+E
 E->(E)
 E->id

Enter Input String :
id+id*id


Stack       Input       Action

$id         +id*id$     SHIFT->id
$E          +id*id$     REDUCE TO E
$E+         id*id$      SHIFT->symbols
```

```
$E+id       *id$        SHIFT->id
$E+E        *id$        REDUCE TO E
$E          *id$        REDUCE TO E
$E*         id$         SHIFT->symbols
$E*id       $           SHIFT->id
$E*E        $           REDUCE TO E
```

Program-5

Question: 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

Program:

[lab5.c]

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

char op[2],arg1[5],arg2[5],result[5];
int 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);
}
```

input.txt:

```
T1 - B = ?
T2 C + D
T3 T1 * T2
A T3 = ?
```

Output:

```
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
```

Program-6

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

Program:

[lab6a.l]

```
%{
#include<stdio.h>
int sl=0;
int ml=0;
%}
%%
"/*"[a-zA-Z0-9' '\t\n]+"*/" ml++;
"//".* sl++;
%%

main()
{
 yyin=fopen("f1.c","r");
 yyout=fopen("f2.c","w");
 yylex();
 fclose(yyin);
 fclose(yyout);
 printf("\n Number Of Single Line Comments Are =
                    %d\n",sl);
 printf("\nNumber Of Multiline Comments Are
                    =%d\n",ml);

}
```

f1.c:

```
#include<stido.h>

int main()

{

 // this is a comment
```

```
 printf("hello");

 /* this is another comment */

}
```

Output:

```
Number Of Single Line Comments Are = 1
Number Of Multi Line Comments Are = 1
```

Program-6

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

Program:

[lab6b.l]

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

%%
[ \t];
[+|-|*|/|=|<|>] {printf("Operator Is
                        %s\n",yytext);return OP;}
[0-9]+ {yylval = atoi(yytext); printf("Number 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("Identifier Is
                        %s\n",yytext);return ID;}
. ;
%%
```

[lab6b.y]

```
%{
#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("inputfile.c", "r");
 if (!myfile)
 {
  printf("I Can't Open Inputfile.c!");
  return -1;
 }
 yyin = myfile;
 do{
  yyparse();
 }while (!feof(yyin));
 printf("Numbers = %d\nKeywords = %d\nIdentifiers =
                    %d\nOperators = %d\n",dig,
                    key,id, op);
}

void yyerror()
{
```

```
 printf("EEK, parse error! Message: ");
 exit(-1);
}
```

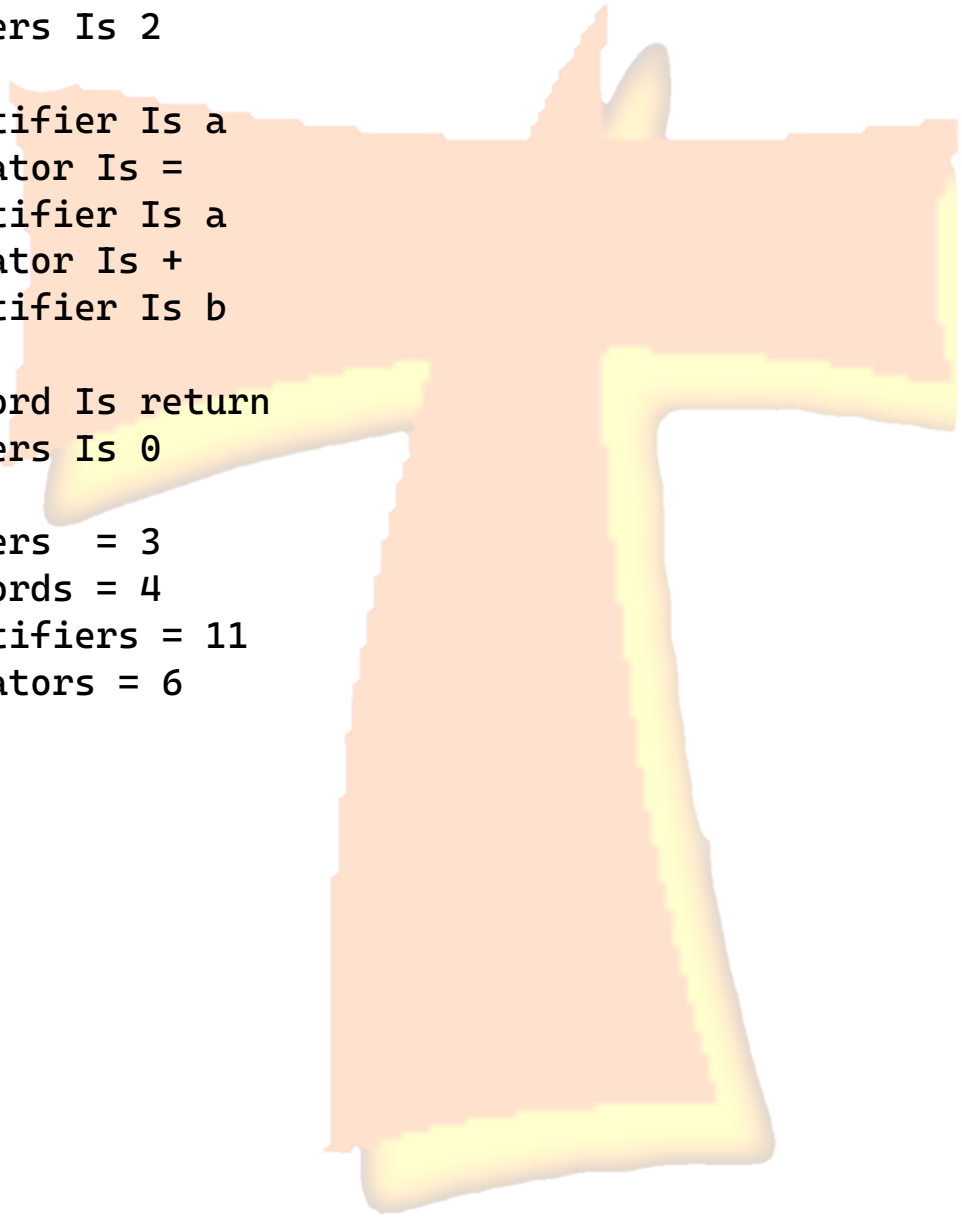
inputfile.txt:

```
#include<stdio.h>

int main()

{

 int a ;

 int b ;

 a = 1 ;

 b = 2 ;

 a = a+b;

 return 0 ;
}
```

Output:

```
Identifier Is include
Operator Is <
Identifer Is stdio
Identifier Is h
Operator Is >

Keyword Is int
Identifier Is main

Keyword Is int
Identifier Is a

Keyword Is int
Identifier Is a
```

```
Identifier Is a
Operator Is =
Numbers Is 1

Identifier Is b
Operator Is =
Numbers Is 2

Identifier Is a
Operator Is =
Identifier Is a
Operator Is +
Identifier Is b

Keyword Is return
Numbers Is 0

Numbers  = 3
Keywords = 4
Identifiers = 11
Operators = 6
```



Program-7

Question: 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.

Program:

[lab7.c]

```
// CPU Scheduling - Round Robin
#include<stdio.h>
struct process
{
 char name;
 int at,bt,wt,tt,rt;
 int completed;
}p[10];

int n;
int q[10];  //queue
int front=-1,rear=-1;
void enqueue(int i)
{
    if(rear==10)
        printf("Overflow");
    rear++;
    q[rear]=i;
    if(front==-1)
        front=0;
}

int dequeue()
{
    if(front==-1)
        printf("Underflow");
    int temp=q[front];
    if(front==rear)
        front=rear=-1;
```

```
    else
        front++;
    return temp;
}

int isInQueue(int i)
{
    int k;
    for(k=front;k<=rear;k++)
    {
        if(q[k]==i)
        return 1;
    }
    return 0;
}

void sortByArrival()
{
    struct process temp;
    int i,j;
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(p[i].at>p[j].at)
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
            }
        }
    }
}
```

```
int main()
{
    int i,j,time=0,sum_bt=0,tq;
    char c;
    float avgwt=0;
     printf("Enter Number Of Processes:\n");
     scanf("%d",&n);
     for(i=0,c='A';i<n;i++,c++)
     {
         p[i].name=c;
         printf("\nProcess %c\n",c);
         printf("\tArrival Time :");
         scanf("%d",&p[i].at);
         printf("\tBurst Time :");
         scanf("%d",&p[i].bt);
         p[i].rt=p[i].bt;
         p[i].completed=0;
         sum_bt+=p[i].bt;
    }
    printf("\nEnter The Time Quantum:");
    scanf("%d",&tq);
    sortByArrival();
    enqueue(0);
    printf("\nProcess Execution Order: ");
    for(time=p[0].at;time<sum_bt;)
    {
        i=dequeue();
        if(p[i].rt<=tq)
        {
            time+=p[i].rt;
            p[i].rt=0;
            p[i].completed=1;
            printf(" %c ",p[i].name);
            p[i].wt=time-p[i].at-p[i].bt;
```

```
            p[i].tt=time-p[i].at;
            for(j=0;j<n;j++)
            {
                if(p[j].at<=time &&
                    p[j].completed!=1&&
                    isInQueue(j)!=1)
                {
                    enqueue(j);
                }
            }
        }
        else
        {
            time+=tq;
            p[i].rt-=tq;
            printf(" %c ",p[i].name);
            for(j=0;j<n;j++)
            {
                if(p[j].at<=time &&
                    p[j].completed!=1&&i!=j&&
                    isInQueue(j)!=1)
                {
                    enqueue(j);
                }
            }
            enqueue(i);
        }
    }
    printf("\n\nName\tArrival Time\tBurst
                    Time\tResponse Time\tTurnAround
                    Time");
    for(i=0;i<n;i++)
    {
        avgwt+=p[i].wt;
```

```
                    printf("\n%c\t\t%d\t\t%d\t\t%d\
                    t\t%d",p[i].name,p[i].at,p[i].b
                    t,p[i].wt,p[i].tt);
    }
    printf("\n\nAverage Waiting Time:%f",avgwt/n);
}
```

Output:

```
Enter Number Of Processes:
4

Process A
        Arrival Time :0
        Burst Time :8

Process B
        Arrival Time :1
        Burst Time :4

Process C
        Arrival Time :12
        Burst Time :9

Process D
        Arrival Time :3
        Burst Time :5

Enter The Time Quantum:4

Process Execution Order: A B C D A C D C
```

Name	Arrival Time	Burst Time	Response Time	TurnAround Time
A	0	8	12	20
B	1	4	3	7
C	2	9	15	24
D	3	5	17	22

Program-8

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

Program:

[lab8.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 = 0;
 printf("Enter The Number Of Processes : ");
 scanf("%d", &p);
 for(i = 0; i< p; i++)
  completed[i] = 0;

 printf("\n\nEnter The Number 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);

  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!!");
```

```
}
```

Output:

```
Enter The Number Of Processes : 3
Enter The Max Matrix For Each Process : 2
For Process 1 : 3 2 1
For Process 2 : 6 1 2
For Process 3 : Enter The Allocation For Each Process :
For Process 1 : 2 0 4
For Process 1 : 3 1 6
For Process 3 : Enter The Available Resources : 1 2 0

Max Matrix:                 Allocation Matrix:
3    2                      2    0
1    6                      4    3
1    2                      1    6

Process 1 Runs To Completion!

Max Matrix:                 Allocation Matrix:
0    0                      0    0
1    6                      4    3
1    2                      1    6

Process 3 Runs To Completion!

Max Matrix:                 Allocation Matrix:
0    0                      0    0
1    6                      4    3
0    0                      0    0

Process 2 Runs To Completion!

The System Is In A Safe State!!
Safe Sequence : < 1 3 2 >
```

Program-9

Question: 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.

Program:

[lab9.c]

```
#include<stdio.h>
#include<stdlib.h>

void FIFO(char [ ],char [ ],int,int);
void lru(char [ ],char [ ],int,int);
void opt(char [ ],char [ ],int,int);

int main()
{
    int ch,YN=1,i,l,f;
    char F[10],s[25];
    printf("\nEnter The Numbr 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]=-1;

    do
    {
        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]=-1;
                    FIFO(s,F,l,f);
                    break;

            case 2: for(i=0;i<f;i++)
                        F[i]=-1;
                    lru(s,F,l,f);
                    break;

            case 3: exit(0);
        }
        printf("\n\nDo You Want To Continue If YES
                    Press 1\nIf NO Press 0 : ");
        scanf("%d",&YN);
    } while(YN==1);
    return(0);
}


void FIFO(char s[],char F[],int l,int f)
{
    int i,j=0,k,flag=0,cnt=0;
    printf("\n\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;
    }
}
```

Output:

```
Enter The Number Of Empty Frames: 3
Enter The Length Of The Strings: 5

********* MENU *********
1:FIFO
2:LRU
3:EXIT
Enter Your Choice: 1

PAGE    FRAMES                  FAULTS
h       h    -    -          Page-Fault0
e       h    e    -          Page-Fault1
l       h    e    l          Page-Fault2
l       h    e    l          No Page-Fault
o       o    e    l          Page-Fault3

Do You Want To Continue If YES Press 1 If NO Press 0 :1

********* MENU *********
1:FIFO
2:LRU
3:EXIT
Enter Your Choice: 2

PAGE    FRAMES                  FAULTS
h       h    -    -          Page-Fault0
e       h    e    -          Page-Fault1
l       h    e    l          Page-Fault2
l       h    e    l          No Page-Fault
```

```
o       e    l    o          Page-Fault3

Do You Want To Continue If YES Press 1 If NO Press 0 :1

********* MENU *********
1:FIFO
2:LRU
3:EXIT
Enter Your Choice: 3
```