

Question 1

Write program to find ϵ – closure of all states of any given NFA with ϵ transition.

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int st;
struct node *link;
};
void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void print_e_closure(int);
static int set[20],nostate,noalpha,s,notransition,c,r,buffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};
void main()
{
int i,j,k,m,t,n;
struct node *temp;
printf("Enter the number of alphabets?\n");
scanf("%d",&noalpha);
getchar();
printf("NOTE:- [ use letter e as epsilon]\n");
printf("NOTE:- [e must be last character ,if it is present]\n");
printf("\nEnter alphabets?\n");
for(i=0;i<noalpha;i++)
{
alphabet[i]=getchar();
getchar();
}
printf("\nEnter the number of states?\n");
scanf("%d",&nostate);
printf("\nEnter no of transition?\n");
scanf("%d",&notransition);
printf("NOTE:- [Transition is in the form-> qno alphabet
qno]\n",notransition);
```

```

printf("NOTE:- [States number must be greater than zero]\n");
printf("\nEnter transition?\n");
for(i=0;i<notransition;i++)
{
scanf("%d %c%d",&r,&c,&s);
insert_trantbl(r,c,s);
}
printf("\n");
printf("e-closure of states.....\n");
printf("-----\n");
for(i=1;i<=nostate;i++)
{
c=0;
for(j=0;j<20;j++)
{
buffer[j]=0;
e_closure[i][j]=0;
}
findclosure(i,i);
printf("\ne-closure(q%d): ",i);
print_e_closure(i);
}
}
void findclosure(int x,int sta)
{
struct node *temp;
int i;
if(buffer[x])
return;
e_closure[sta][c++]=x;
buffer[x]=1;
if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL)
{
temp=transition[x][noalpha-1];
while(temp!=NULL)
{
findclosure(temp->st,sta);
temp=temp->link;
}
}
}
void insert_trantbl(int r,char c,int s)
{
int j;
struct node *temp;
j=findalpha(c);
if(j==999)

```

```

{
printf("error\n");
exit(0);
}
temp=(struct node *)malloc(sizeof(struct node));
temp->st=s;
temp->link=transition[r][j];
transition[r][j]=temp;
}
int findalpha(char c)
{
int i;
for(i=0;i<noalpha;i++)
if(alphabet[i]==c)
return i;
return(999);
}
void print_e_closure(int i)
{
int j;
printf("{");
for(j=0;e_closure[i][j]!=0;j++)
printf("q%d,",e_closure[i][j]);
printf("}");
}

```

Output :

```

Enter the number of states: 4
Enter the number of alphabets: 2
Enter the alphabets:
(Enter e for Epsilon & add it as the last alphabet)
a
e
Enter the number of transistions: 4

NOTE: State number begins at 1
Enter transition in the format: CurrentStateNumber Alphabet NextStateNumber
1 e 2
2 a 3
1 e 3
3 e 4
Epsilon closure for the states are:
e-closure(q1) = { q1, q3, q4, q2 }
e-closure(q2) = { q2 }
e-closure(q3) = { q3, q4 }
e-closure(q4) = { q4 }

```

Question 2

Write program to convert NFA with ϵ transition to NFA without ϵ transition.

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int st;
    struct node *link;
};

void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void findfinalstate(void);
void unionclosure(int);
void print_e_closure(int);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};
void main()
{
    int i,j,k,m,t,n;

    struct node *temp;
    printf("enter the number of alphabets?\n");
    scanf("%d",&noalpha);
    getchar();
    printf("NOTE:- [ use letter e as epsilon]\n");

    printf("NOTE:- [e must be last character ,if it is present]\n");

    printf("\nEnter alphabets?\n");
    for(i=0;i<noalpha;i++)
    {
        alphabet[i]=getchar();
        getchar();
    }
}
```

```

    }
    printf("Enter the number of states?\n");
    scanf("%d",&nostate);
    printf("Enter the start state?\n");
    scanf("%d",&start);
    printf("Enter the number of final states?\n");
    scanf("%d",&nofinal);
    printf("Enter the final states?\n");
    for(i=0;i<nofinal;i++)
        scanf("%d",&finalstate[i]);
    printf("Enter no of transition?\n");
    scanf("%d",&notransition);
    printf("NOTE:- [Transition is in the form--> qno  alphabet
qno]\n",notransition);
    printf("NOTE:- [States number must be greater than zero]\n");
    printf("\nEnter transition?\n");
    for(i=0;i<notransition;i++)
    {

        scanf("%d %c%d",&r,&c,&s);
        insert_trantbl(r,c,s);

    }

    printf("\n");

    for(i=1;i<=nostate;i++)
    {
        c=0;
        for(j=0;j<20;j++)

        {
            buffer[j]=0;
            e_closure[i][j]=0;
        }
        findclosure(i,i);
    }
    printf("Equivalent NFA without epsilon\n");
    printf("-----\n");
    printf("start state:");
    print_e_closure(start);
    printf("\nAlphabets:");
    for(i=0;i<noalpha;i++)
        printf("%c ",alphabet[i]);
    printf("\n States :" );
    for(i=1;i<=nostate;i++)

```

```

        print_e_closure(i);

printf("\nTnransitions are...\n");

for(i=1;i<=nostate;i++)
{
    for(j=0;j<noalpha-1;j++)
    {
        for(m=1;m<=nostate;m++)
            set[m]=0;
        for(k=0;e_closure[i][k]!=0;k++)
        {
            t=e_closure[i][k];
            temp=transition[t][j];
            while(temp!=NULL)
            {
                unionclosure(temp->st);
                temp=temp->link;
            }
        }
        printf("\n");
        print_e_closure(i);
        printf("%c\t",alphabet[j] );
        printf("{ ");
        for(n=1;n<=nostate;n++)
        {
            if(set[n]!=0)
                printf("q%d,",n);
        }
        printf("}");
    }
}
printf("\n Final states:");
findfinalstate();

}

void findclosure(int x,int sta)
{
    struct node *temp;
    int i;

```

```

        if(buffer[x])
            return;
        e_closure[sta][c++]=x;
        buffer[x]=1;
        if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL)
        {
            temp=transition[x][noalpha-1];
            while(temp!=NULL)
            {
                findclosure(temp->st,sta);
                temp=temp->link;
            }
        }
    }

void insert_trantbl(int r,char c,int s)
{
    int j;
    struct node *temp;
    j=findalpha(c);
    if(j==999)
    {
        printf("error\n");
        exit(0);
    }
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

int findalpha(char c)
{
    int i;
    for(i=0;i<noalpha;i++)
        if(alphabet[i]==c)
            return i;

    return(999);
}

void unionclosure(int i)
{
    int j=0,k;
    while(e_closure[i][j]!=0)

```

```

        {
            k=e_closure[i][j];
            set[k]=1;
            j++;
        }
    }
void findfinalstate()
{
    int i,j,k,t;
    for(i=0;i<nofinal;i++)
    {
        for(j=1;j<=nostate;j++)
        {
            for(k=0;e_closure[j][k]!=0;k++)
            {
                if(e_closure[j][k]==finalstate[i])
                {
                    print_e_closure(j);
                }
            }
        }
    }
}

}

void print_e_closure(int i)
{
    int j;
    printf("{");
    for(j=0;e_closure[i][j]!=0;j++)
        printf("q%d,",e_closure[i][j]);
    printf("}\t");
}

```


Output :

```
main.c:56:28: warning: format '%c' expects argument of type 'char *', but argument 3 has type 'int *' [-Wformat=]
   56 |             scanf("%d %c%d", &r, &c, &s);
      |             ^~          ~~
      |             |          |
      |             char *    int *
      |             %lc
enter the number of alphabets?
4
NOTE:- [ use letter e as epsilon]
NOTE:- [e must be last character ,if it is present]

Enter alphabets?
a
b
c
e
Enter the number of states?
3
Enter the start state?
1
Enter the number of final states?
1
Enter the final states?
3
Enter no of transition?
5
NOTE:- [Transition is in the form--> qno  alphabet  qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1  a  1
1  e  2
2  b  2
2  e  3
3  c  3

Equivalent NFA without epsilon
-----
start state:{q1,q2,q3,}
Alphabets:a b c e
States :{q1,q2,q3,}    {q2,q3,}    {q3,}
Transitions are...:

{q1,q2,q3,}    a    {q1,q2,q3,}
{q1,q2,q3,}    b    {q2,q3,}
{q1,q2,q3,}    c    {q3,}
{q2,q3,}       a    {}
{q2,q3,}       b    {q2,q3,}
{q2,q3,}       c    {q3,}
{q3,}          a    {}
{q3,}          b    {}
{q3,}          c    {q3,}
Final states:{q1,q2,q3,}    {q2,q3,}    {q3,}

...Program finished with exit code 0
Press ENTER to exit console.
```

Question 3

Write program to convert NFA to DFA

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int st;
    struct node *link;
};
struct node1
{
    int nst[20];
};

void insert(int ,char, int);
int findalpha(char);
void findfinalstate(void);
int insertdfastate(struct node1);
int compare(struct node1,struct node1);
void printnewstate(struct node1);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,buffer[20];
int complete=-1;
char alphabet[20];
static int eclosure[20][20]={0};
struct node1 hash[20];
struct node * transition[20][20]={NULL};
void main()
{
    int i,j,k,m,t,n,l;
    struct node *temp;
    struct node1 newstate={0},tmpstate={0};

    printf("Enter the number of alphabets?\n");
    printf("NOTE:- [ use letter e as epsilon]\n");
    printf("NOTE:- [e must be last character ,if it is present]\n");
    printf("\nEnter No of alphabets?\n");
    scanf("%d",&noalpha);
```

```

getchar();
for(i=0;i<noalpha;i++)
{

alphabet[i]=getchar();
getchar();
}
printf("Enter the number of states?\n");
scanf("%d",&nostate);
printf("Enter the start state?\n");
scanf("%d",&start);
printf("Enter the number of final states?\n");
scanf("%d",&nofinal);
printf("Enter the final states?\n");
for(i=0;i<nofinal;i++)
scanf("%d",&finalstate[i]);
printf("Enter no of transition?\n");

scanf("%d",&notransition);
printf("NOTE:- [Transition is in the form-> qno alphabet qno]\n",notransition);
printf("NOTE:- [States number must be greater than zero]\n");
printf("\nEnter transition?\n");

for(i=0;i<notransition;i++)
{

scanf("%d %c%d",&r,&c,&s);
insert(r,c,s);

}
for(i=0;i<20;i++)
{
for(j=0;j<20;j++)
hash[i].nst[j]=0;
}
complete=-1;
i=-1;
printf("\nEquivalent DFA.....\n");
printf(".....\n");

printf("Trnsitions of DFA\n");

newstate.nst[start]=start;
insertdfastate(newstate);
while(i!=complete)

```

```

{
    i++;
    newstate=hash[i];
    for(k=0;k<noalpha;k++)
    {
        c=0;
        for(j=1;j<=nostate;j++)
            set[j]=0;
        for(j=1;j<=nostate;j++)
        {
            l=newstate.nst[j];
            if(l!=0)
            {
                temp=transition[l][k];
                while(temp!=NULL)
                {
                    if(set[temp->st]==0)
                    {
                        c++;
                        set[temp->st]=temp->st;
                    }
                    temp=temp->link;
                }
            }
        }
        printf("\n");
        if(c!=0)
        {
            for(m=1;m<=nostate;m++)
                tmpstate.nst[m]=set[m];

            insertdfastate(tmpstate);

            printnewstate(newstate);
            printf("%c\t",alphabet[k]);
            printnewstate(tmpstate);
            printf("\n");
        }
        else
        {
            printnewstate(newstate);
            printf("%c\t", alphabet[k]);
            printf("NULL\n");
        }
    }
}

```

```

    }
    }
    printf("\nStates of DFA:\n");
    for(i=0;i<=complete;i++)
    printnewstate(hash[i]);
    printf("\n Alphabets:\n");
    for(i=0;i<noalpha;i++)
    printf("%c\t",alphabet[i]);
    printf("\n Start State:\n");
    printf("q%d",start);
    printf("\nFinal states:\n");
    findfinalstate();

}

int insertdfastate(struct node1 newstate)
{
    int i;
    for(i=0;i<=complete;i++)
    {
        if(compare(hash[i],newstate))
            return 0;
    }
    complete++;
    hash[complete]=newstate;
    return 1;
}

int compare(struct node1 a,struct node1 b)
{
    int i;

    for(i=1;i<=nostate;i++)
    {
        if(a.nst[i]!=b.nst[i])
            return 0;
    }
    return 1;

}

void insert(int r,char c,int s)
{
    int j;
    struct node *temp;
    j=findalpha(c);
    if(j==999)

```

```

    {
    printf("error\n");
    exit(0);
    }
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

```

```

int findalpha(char c)
{
    int i;
    for(i=0;i<noalpha;i++)
    if(alphabet[i]==c)
    return i;

    return(999);

}

```

```

void findfinalstate()
{
    int i,j,k,t;

    for(i=0;i<=complete;i++)
    {
        for(j=1;j<=nostate;j++)
        {
            for(k=0;k<nofinal;k++)
            {
                if(hash[i].nst[j]==finalstate[k])
                {
                    printnewstate(hash[i]);
                    printf("\t");
                    j=nostate;
                    break;
                }
            }
        }
    }
}

```

```

void printnewstate(struct node1 state)

```

```
{
int j;
printf("{ ");
for(j=1;j<=nostate;j++)
{
if(state.nst[j]!=0)
printf("q%d,",state.nst[j]);
}
printf("}\t");
}
```

Output :

```
Enter the start state?
1
Enter the number of final states?
2
Enter the final states?
3
4
Enter no of transition?
8
NOTE:- [Transition is in the form-> qno alphabet qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 b 1
1 a 2
2 b 2
2 a 3
3 a 4
3 b 4
4 b 3

Equivalent DFA.....
.....
Transitions of DFA

{q1,}    a      {q1,q2,}
{q1,}    b      {q1,}

{q1,q2,}    a      {q1,q2,q3,}
{q1,q2,}    b      {q1,q2,}

{q1,q2,q3,}    a      {q1,q2,q3,q4,}
{q1,q2,q3,}    b      {q1,q2,q4,}

{q1,q2,q3,q4,}    a      {q1,q2,q3,q4,}
{q1,q2,q3,q4,}    b      {q1,q2,q3,q4,}

{q1,q2,q4,}    a      {q1,q2,q3,}
{q1,q2,q4,}    b      {q1,q2,q3,}

States of DFA:
{q1,}    {q1,q2,}    {q1,q2,q3,}    {q1,q2,q3,q4,}    {q1,q2,q4,}
Alphabets:
a      b
Start State:
q1
Final states:
{q1,q2,q3,}    {q1,q2,q3,q4,}    {q1,q2,q4,}
```


Question 4

Write program to minimize any given DFA.

Program

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

static int nostate,noalpha,s,notransition,nofinal,start,finalstate[20],r;
char alphabet[20];
int transition_map[30][30], table[30][30], nonfinalstate[20], partition[20][20];

int findalpha(char a)
{
    int i;
    for(i=0;i<noalpha;i++)
        if(alphabet[i]==a)
            return i;
    return(-1);
}

int main() {
    int i,j,p[20],q[20],k;
    char a;

    for(i=0;i<30;i++){
        for(j=0;j<30;j++)
            transition_map[i][j]=-1;
    }
    printf("Enter the number of alphabets: ");
    scanf("%d",&noalpha);
    getchar();
    printf("Enter the alphabets: \n");
    for(i=0;i<noalpha;i++)
    {
        alphabet[i]=getchar();
        getchar();
    }
    printf("Enter the number of states: ");
    scanf("%d",&nostate);
    printf("Enter the start state: ");
    scanf("%d",&start);
    printf("Enter the number of final states: ");
```

```

scanf("%d",&nofinal);
printf("Enter the final states:\n");
for(i=0;i<nofinal;i++)
    scanf("%d",&finalstate[i]);
printf("Enter no of transition: ");
scanf("%d",&notransition);
printf("Enter Transition in the form -> state alphabet next_state\n");
for(i=0;i<notransition;i++)
{
    scanf("%d %c %d",&r,&a,&s);
    j=findalpha(a);
    if (j==-1){printf("\nerror\n"); exit(1);}
    transition_map[r][j] = s;
}
for(i=0;i<nostate;i++){
    for(j=0;j<i;j++){
        table[i][j]=0;
    }
}

int f=0;
k=0;
for(i=0;i<nostate;i++){
    f=0;
    for(j=0;j<nofinal;j++){
        if(i==finalstate[j])
            { f=1;break;}
    }
    if(f==0){nonfinalstate[k++]=i;}
}

for(i=0;i<nofinal;i++){
    for(j=0;j<(nostate-nofinal);j++)
        if(nonfinalstate[j]>finalstate[i])
            table[nonfinalstate[j]][finalstate[i]]=1;
    else
        table[finalstate[i]][nonfinalstate[j]]=1;
}
int change = 1;
while(change==1){
    change=0;
    for(i=0;i<nostate;i++){
        for(j=0;j<i;j++){
            if(table[i][j]!=1){
                for(k=0;k<noalpha;k++)
                    p[k]=transition_map[i][k];
                for(k=0;k<noalpha;k++)

```

```

        q[k]=transition_map[j][k];
        for(k=0;k<noalpha;k++){
            if(p[k]>q[k]){
                if (table[p[k]][q[k]]==1){
                    change=1;
                    table[i][j]=1;
                    break;
                }
            }
            else if(p[k]<q[k]){
                if (table[q[k]][p[k]]==1){
                    change=1;
                    table[i][j]=1;
                    break;
                }
            }
        }
    }
}

k=0;
for(i=0;i<nostate;i++){
    k=0;
    partition[i][k++]=i;
    for(j=0;j<i;j++){
        if(table[i][j]==0){
            partition[i][k++]=j;
        }
        partition[i][k]=-1;
    }
}
int newstate[20]={0},m;

printf("\nStates in minimized DFA");
printf("\n-----\n");
for(i=nostate-1;i>=0;i--){
    k=0;
    if(newstate[i]==0){
        printf(" ");
        while(partition[i][k]!=-1){
            if(newstate[partition[i][k]]==0){
                newstate[partition[i][k]]=1;
                printf("q%d ",partition[i][k]);
            }
            k++;
        }
        printf("}\n");
    }
}

```

```

    }
}
return 0;
}

```

Output :

```

Enter the number of alphabets: 2
Enter the alphabets:
a
b
Enter the number of states: 6
Enter the start state: 0
Enter the number of final states: 3
Enter the final states:
1 2 4
Enter no of transition: 12
Enter Transition in the form -> state alphabet next_state
0 a 3
0 b 1
1 a 2
1 b 5
2 a 2
2 b 5
3 a 0
3 b 4
4 a 2
4 b 5
5 a 5
5 b 5

States in minimized DFA
-----
{q5 }
{q4 q1 q2 }
{q3 q0 }

```

Question 1

Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and newlines.

Program :

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

int isKeyword(char buffer[]){
    char keywords[32][10] =
{"auto","break","case","char","const","continue","default",
    "do","double","else","enum","extern","float","for",
    "goto",
    "if","int","long","register","return","short","sig
ned",
    "sizeof","static","struct","switch","typedef","uni
on",
    "unsigned","void","volatile","while"};

    int i;
    for(i = 0; i < 32; ++i){
        if(strcmp(keywords[i], buffer) == 0){
            return 1;
        }
    }
    return 0;
}

int main() {
    char c, buffer[31], operators[] = "+-*/%=";
    FILE *fp;
    int i, j=0;

    fp = fopen("Program","r");

    if(fp == NULL){
        printf("Error while opening the file\n");
        exit(0);
    }

    while((c = fgetc(fp)) != EOF) {
        for(i = 0; i < 6; ++i){
            if(c == operators[i])
                printf("%c is operator\n", c);
        }
    }
```

```

        if(isalnum(c)) {
            buffer[j++] = c;
        } else if((c == ' ' || c == '\t' || c == '\n') && (j != 0)) {
            buffer[j] = '\0';
            j = 0;

            if(isKeyword(buffer) == 1)
                printf("%s is keyword\n", buffer);
            else
                printf("%s is identifier\n", buffer);
        }
    }
    fclose(fp);
    return 0;
}

```

Input :

```

Pg1 > ≡ Program
1  void main()
2  {
3      int a = 5;
4      int b = 10;
5      int c = a + b;
6  }

```

Output :

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg1>gcc lexanalyzer.c
```

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg1>a
```

```
void is keyword  
main is identifier  
int is keyword  
a is identifier  
a is identifier  
= is operator  
int is keyword  
a is identifier  
= is operator  
5 is identifier  
int is keyword  
b is identifier  
= is operator  
10 is identifier  
int is keyword  
c is identifier  
= is operator  
a is identifier  
+ is operator  
b is identifier
```

Question 2

Write a lex program to recognize all strings which does not contain first four characters of your name as a substring.

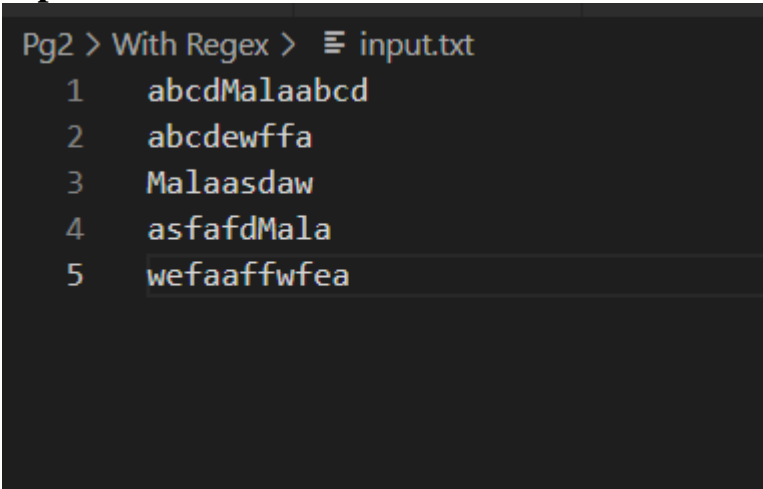
Program.l:

```
%%  
.*Mala.*\n {fprintf(yyout, "");}  
. { fprintf(yyout, "%s", yytext); }  
%%
```

```
int yywrap(){ }
```

```
int main()  
{  
    extern FILE *yyin, *yyout;  
  
    yyin = fopen("input.txt", "r");  
  
    yyout = fopen("output.txt", "w");  
  
    yylex();  
    return 0;  
}
```

Input:



```
Pg2 > With Regex > ≡ input.txt  
1   abcdMalaabcd  
2   abcdewffa  
3   Malaasdaw  
4   asfafdMala  
5   wefaaffwfea
```

Output :


```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg2\With Regex>flex pg1.1
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg2\With Regex>gcc lex.yy.c
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg2\With Regex>a
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg2\With Regex>type output.txt
abcdMalaabcd
abcdewffa
Malaasdaw
asfafdMala
wefaaffwfea
```

Question 3

Write a YACC program to recognize a valid variable which starts with a letter followed by any number of letters or digits.

Identifier.l

```
% {  
  
    #include "y.tab.h"  
  
% }  
  
%%  
  
[a-zA-Z_][a-zA-Z_0-9]* return letter;  
  
[0-9]          return digit;  
  
.              return yytext[0];  
  
\n            return 0;  
  
%%  
  
int yywrap()  
  
{  
  
return 1;  
  
}
```

Identifier.y :

```
%{  
  
    #include<stdio.h>  
  
    int valid=1;  
  
}%  
  
%token digit letter  
  
%%  
  
start : letter s  
  
s :    letter s  
      | digit s  
      |  
      ;  
  
%%  
  
int yyerror()  
{  
    printf("\nIts not a identifier!\n");  
    valid=0;  
    return 0;  
}  
  
int main()  
{  
    printf("\nEnter a name to tested for identifier ");  
  
    yyparse();
```

```
if(valid)

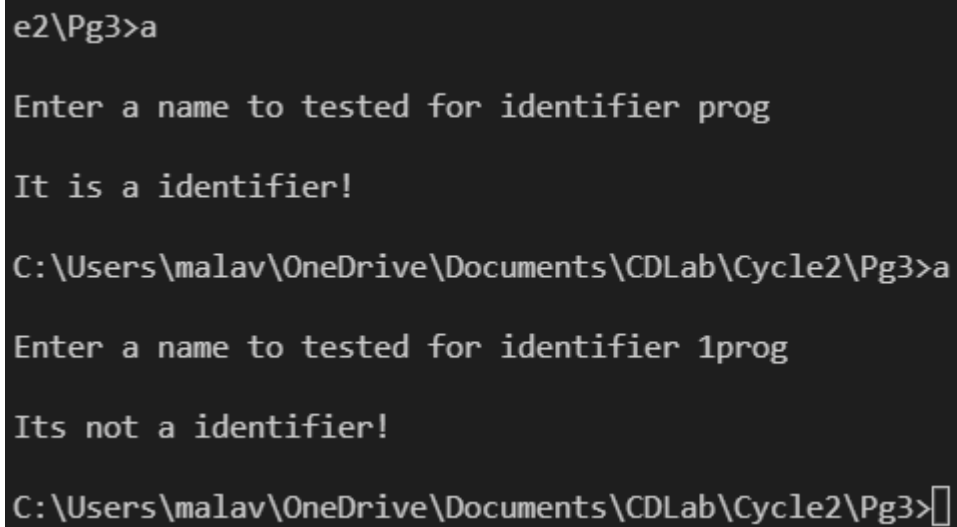
{

    printf("\nIt is a identifier!\n");

}

}
```

Output :



```
e2\Pg3>a

Enter a name to tested for identifier prog

It is a identifier!

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg3>a

Enter a name to tested for identifier 1prog

Its not a identifier!

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg3>
```

Question 2

Implementation of Calculator using LEX and YACC

Program :

Lex:

```
% {  
  
#include<stdio.h>  
  
#include "y.tab.h"  
  
extern int yyval;  
  
% }  
  
%%  
  
[0-9]+ {  
    yyval=atoi(yytext);  
    return NUMBER;  
}  
  
[\t] ;  
  
[\n] return 0;  
  
. return yytext[0];  
  
%%  
  
int yywrap()  
  
{  
  
return 1;
```

```
}
```

Yacc :

```
% {
```

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

```
    int flag=0;
```

```
% }
```

```
%token NUMBER
```

```
%left '+' '-'
```

```
%left '*' '/' '%'
```

```
%left '(' ')'
```

```
%%
```

```
ArithmeticExpression: E{
```

```
    printf("\nResult=%d\n",$$);
```

```
    return 0;
```

```
};
```

```
E:E+'E' {$$=$1+$3;}
```

```
|E-'E' {$$=$1-$3;}
```

```
|E'*'E {$$=$1*$3;}
```

```
|E/'E' {$$=$1/$3;}
```

```
|E%'E' {$$=$1%$3;}
```

```
|'('E')' {$$=$2;}
```

```
| NUMBER {$$=$1;}
```

```
;
```

```
%%
```

```
void main()
```

```
{
```

```
    printf("\nEnter Any Arithmetic Expression which can have operations  
Addition, Subtraction, Multiplication, Division, Modulus and Round  
brackets:\n");
```

```
    yyparse();
```

```
    if(flag==0)
```

```
        printf("\nEnter arithmetic expression is Valid\n\n");
```

```
}
```

```
void yyerror()
```

```
{
```

```
    printf("\nEnter arithmetic expression is Invalid\n\n");
```

```
    flag=1;
```

```
}
```

Output :

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>bison -dy
variable.y
bison: cannot open file `variable.y': No such file or direct
ory

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>bison -dy
calc.y

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>flex calc
.l

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>gcc lex.y
y.c y.tab.c
```

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>a

Enter Any Arithmetic Expression which can have operations Ad
dition, Subtraction, Multiplication, Divison, Modulus and Ro
und brackets:
4+5

Result=9

Entered arithmetic expression is Valid

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>a

Enter Any Arithmetic Expression which can have operations Ad
dition, Subtraction, Multiplication, Divison, Modulus and Ro
und brackets:
5/10

Result=0

Entered arithmetic expression is Valid

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg5>a

Enter Any Arithmetic Expression which can have operations Ad
dition, Subtraction, Multiplication, Divison, Modulus and Ro
und brackets:
5+

Entered arithmetic expression is Invalid
```


Question 5

Convert the BNF rules into YACC form and write code to generate abstract syntax tree

Program:

Yacc:

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

int yylex();
int yyerror();

struct quad
{
    char op[5];
    char arg1[10];
    char arg2[10];
    char result[10];
} QUAD[30];

struct stack
{
    int items[100];
    int top;
} stk;

int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
% }

%union
{
char var[10];
}

%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
```

```
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST
WHILELOOP
%left '-' '+'
%left '*' '/'
```

```
% %
```

```
PROGRAM : MAIN BLOCK
```

```
;
```

```
BLOCK: '{' CODE '}'
```

```
;
```

```
CODE: BLOCK
```

```
| STATEMENT CODE
```

```
| STATEMENT
```

```
;
```

```
STATEMENT: DESCT ';' | ASSIGNMENT ';' | CONDST
```

```
| WHILEST
```

```
;
```

```
DESCT: TYPE VARLIST
```

```
;
```

```
VARLIST: VAR ',' VARLIST
```

```
| VAR
```

```
;
```

```
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
}
```

```
;
```

```
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);} | EXPR '-' EXPR {AddQuadruple("-",
```

```
$1,$3,$$);} | EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);} | EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);} | '-' EXPR {AddQuadruple("UMIN",$2,"",
```

```
$$);} | '(' EXPR ')' {strcpy($$, $2);} | VAR
```

```
| NUM
```

```
;
```

```
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
```

```

printf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{

tInd=pop();
Ind=pop();
push(tInd);
printf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
printf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
printf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
printf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);

```

```

strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;

%%

```

```

extern FILE *yyin;
int main(int argc,char *argv[])
{
    FILE *fp;
    int i;
    if(argc>1)
    {
        fp=fopen(argv[1],"r");
        if(!fp)
        {
            printf("\n File not found");
            exit(0);
        }
        yyin=fp;
    }
    yyparse();
    printf("\n\n\t\t ----- \n\t\t Pos Operator Arg1 Arg2
Result\n\t\t -----");
    for(i=0;i<Index;i++)
    {
        printf("\n\t\t %d\t %s\t %s\t
%s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
    }
    printf("\n\t\t -----");
    printf("\n\n");
    return 0;
}

void push(int data)
{

```

```

    stk.top++;
    if(stk.top==100)
    {
        printf("\n Stack overflow\n");
        exit(0);
    }
    stk.items[stk.top]=data;
}

int pop()
{
    int data;
    if(stk.top== -1)
    {
        printf("\n Stack underflow\n");
        exit(0);
    }
    data=stk.items[stk.top--];
    return data;
}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
    strcpy(QUAD[Index].op,op);
    strcpy(QUAD[Index].arg1,arg1);
    strcpy(QUAD[Index].arg2,arg2);
    sprintf(QUAD[Index].result,"t%d",tIndex++);
    strcpy(result,QUAD[Index++].result);
}

yyerror()
{
    printf("\n Error on line no:%d",LineNo);
}

```

Lex :

```

%{
#include "y.tab.h"
#include <stdio.h>
#include <string.h>
int LineNo=1;
%}

identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)

```

%%

```
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
```

%%

```
int yywrap(void){ };
```

Input:

```
1  main()
2  {
3      int a, b, c;
4      if (a < b)
5      {
6          a = a + b;
7      }
8      while (a < b)
9      {
10         a = a + b;
11     }
12     if (a <= b)
13     {
14         c = a - b;
15     }
16     else
17     {
18         c = a + b;
19     }
20 }
```

Output :

C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg6>a input.c

```
-----
Pos Operator Arg1 Arg2 Result
-----
0      <      a      b      t0
1      ==     t0     FALSE  5
2      +      a      b      t1
3      =      t1
4      GOTO
5      <      a      b      t2
6      ==     t2     FALSE 10
11     ==     t4     FALSE 15
12     -      a      b      t5
13     =      t5
14     GOTO
15     +      a      b      t6
16     =      t6      c
-----
```

Question 6

Write a YACC program to check the syntax of FOR statement in C

Program :

Lex :

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

%%

for                return FOR;
\< |
\<= |
== |
\>= |
\> |
!=                return RELOP;
\+ \+             return INC;
--               return DEC;
\\| |
&&               return LOGOP;
[a-zA-Z_][a-zA-Z_0-9]* return ID;
[0-9]+           return NUM;
[\t\n ]         ;
.               return yytext[0];
%%

int yywrap() { }
```

Yacc :

```
% {
    #include <stdio.h>
    #include <stdlib.h>
    extern FILE *yyin;
% }

%token FOR RELOP LOGOP ID NUM INC DEC;
%right '='
```



```
%left LOGOP
%left RELOP
%left '+' '-'
%left '*' '/'
%left INC DEC
```

```
%%
start:      st                      { printf("Input Accepted!\n");
exit(0); }
st:         FOR '(' loop_expr ';' condition ';' loop_expr ')' block ;
loop_expr:  E | ;
block:      '{' code '}' | statement ;
code:       statement code | statement ;
statement:  E ';' | st ;
```

```
E:         ID '=' E
          | E '+' E
          | E '-' E
          | E '*' E
          | E '/' E
          | E RELOP E
          | E LOGOP E
          | ID INC
          | ID DEC
          | ID
          | NUM
          ;
```

```
condition:  E RELOP E
          | E LOGOP E
          ;
```

```
%%
```

```
int yyerror(char* s) {
    fprintf(stderr, "%s\n", s);
}
```

```
int main(int argc, char* argv[]) {
    FILE *fp;
    if(argc>1) {
        fp=fopen(argv[1],"r");
        if(!fp) {
            perror(fp);
            printf("%s not found", argv[1]);
            exit(0);
        }
    }
}
```

```
        yyin=fp;
    }
    printf("Checking the validity of for loop...\n");
    yyparse();
}
```

Input:

```
1  for (i=0; i < 100 ; i++) {
2      temp = i + 10;
3      a = b + c;
4  }
```

Output :

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle2\Pg7>a input.c
Checking the validity of for loop...
Input Accepted!
```

Question 1

Develop an operator precedence parser for a given language.

Program :

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

#define MAX_SIZE 20

void main() {
    int numOfTerminals, top = -1, i, j, k, row, col;
    char terminals[10], operatorPrecedence[10][10], stack[MAX_SIZE],
    input[MAX_SIZE];
    printf("Enter the no. of terminals: ");
    scanf("%d", &numOfTerminals);
    printf("Enter the terminals: \n");
    scanf("%s", terminals);

    //Operator precedence table
    printf("Enter the operator precedence table values: ");
    for(i = 0; i < numOfTerminals; i++)
        for(j = 0; j < numOfTerminals; j++) {
            printf("\nEnter the precedence value for %c %c: ",
terminals[i], terminals[j]);
            scanf(" %c", &operatorPrecedence[i][j]);
        }

    printf("\n-----OPERATOR PRECEDENCE TABLE-----\n");
    for(i = 0; i < numOfTerminals; i++)
        printf("\t%c", terminals[i]);
    for(i = 0; i < numOfTerminals; i++) {
        printf("\n%c", terminals[i]);
        for(j = 0; j < numOfTerminals; j++) {
            printf("\t%c", operatorPrecedence[i][j]);
        }
    }

    //Parse input
    stack[++top] = '$';
    printf("\nInput the string to parse: ");
    scanf("%s", input);

    i = 0;
    printf("\nSTACK\t\tINPUT STRING\tACTION\n");
    printf("%s\t\t%s\t\t", stack, input);

    while(i <= strlen(input)) {
        for(k = 0; k < numOfTerminals; k++) {
            if(stack[top] == terminals[k]) {
                row = k;
                break;
            }
        }
    }
}
```

```

    }
    for(k = 0; k < numOfTerminals; k++) {
        if(input[i] == terminals[k]) {
            col = k;
            break;
        }
    }

    if((stack[top] == '$') && (input[i] == '$')) {
        printf("String is accepted!\n");
        break;
    }

    if((operatorPrecedence[row][col] == '<') ||
(operatorPrecedence[row][col] == '=')) {
        stack[++top] = operatorPrecedence[row][col];
        stack[++top] = input[i];
        printf("SHIFT %c", input[i]);
        i++;
    } else if(operatorPrecedence[row][col] == '>') {
        while(stack[top] != '<')
            --top;
        --top;
        printf("REDUCE");
    } else {
        printf("String is not accepted\n");
        break;
    }

    printf("\n");
    for(k = 0; k <= top; k++)
        printf("%c", stack[k]);

    printf("\t\t");
    for(k = i; k < strlen(input); k++)
        printf("%c", input[k]);
    printf("\t\t");
}
}

```

Output :

```
File Edit View Search Terminal Help
mec@cll-1-1:~/CS78$ touch pgl.c
mec@cll-1-1:~/CS78$ gcc pgl.c
mec@cll-1-1:~/CS78$ ./a.out
Enter the no. of terminals: 4
Enter the terminals:
+*i$
Enter the operator precedence table values:
Enter the precedence value for + +: >

Enter the precedence value for + *: <

Enter the precedence value for + i: <

Enter the precedence value for + $: >

Enter the precedence value for * +: >

Enter the precedence value for * *: >

Enter the precedence value for * i: <

Enter the precedence value for * $: >

Enter the precedence value for i +: >

Enter the precedence value for i *: >

Enter the precedence value for i i: =

Enter the precedence value for i $: >

Enter the precedence value for $ +: <

Enter the precedence value for $ *: <

Enter the precedence value for $ i: <

Enter the precedence value for $ $: A

-----OPERATOR PRECEDENCE TABLE-----
+      +      *      i      $
+      >      <      <      >
*      >      >      <      >
i      >      >      =      >
$      <      <      <      A
Input the string to parse: i*i+i$
```

Enter the precedence value for \$ \$: A

```
-----OPERATOR PRECEDENCE TABLE-----
+      +      *      i      $
+      >      <      <      >
*      >      >      <      >
i      >      >      =      >
$      <      <      <      A
Input the string to parse: i*i+i$
```

STACK	INPUT STRING	ACTION
\$	i*i+i\$	SHIFT i
\$<i	*i+i\$	REDUCE
\$	*i+i\$	SHIFT *
\$<*	i+i\$	SHIFT i
\$<+<i	+i\$	REDUCE
\$<*	+i\$	REDUCE
\$	+i\$	SHIFT +
\$<+	i\$	SHIFT i
\$<+<i	\$	REDUCE
\$<+	\$	REDUCE
\$	\$	String is accepted!

Question 2

Write program to find Simulate First and Follow of any given grammar.

Program :

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
int n, m = 0, p, i = 0, j = 0;
char a[10][10], f[10];
void follow(char c);
void first(char c);
int main()
{
    int i, z;
    char c, ch;
    //clrscr();
    printf("Enter the no of productions:\n");
    scanf("%d", &n);
    printf("Enter the productions:\n");
    for (i = 0; i < n; i++)
        scanf("%s%c", a[i], &ch);
    do
    {
        m = 0;
        printf("Enter the elemets whose fisrt & follow is to be found:");
        scanf("%c", &c);
        first(c);
        printf("First(%c)={", c);
        for (i = 0; i < m; i++)
            printf("%c", f[i]);
        printf("}\n");
        strcpy(f, " ");
        //flushall();
        m = 0;
        follow(c);
        printf("Follow(%c)={", c);
        for (i = 0; i < m; i++)
            printf("%c", f[i]);
        printf("}\n");
        printf("Continue(0/1)?");
        scanf("%d%c", &z, &ch);
    } while (z == 1);
    return (0);
}

void first(char c)
{
    int k;
```

```

    if (!isupper(c))
        f[m++] = c;
    for (k = 0; k < n; k++)
    {
        if (a[k][0] == c)
        {
            if (a[k][2] == '$')
                follow(a[k][0]);
            else if (islower(a[k][2]))
                f[m++] = a[k][2];
            else
                first(a[k][2]);
        }
    }
}

void follow(char c)
{
    if (a[0][0] == c)
        f[m++] = '$';
    for (i = 0; i < n; i++)
    {
        for (j = 2; j < strlen(a[i]); j++)
        {
            if (a[i][j] == c)
            {
                if (a[i][j + 1] != '\0')
                    first(a[i][j + 1]);
                if (a[i][j + 1] == '\0' && c != a[i][0])
                    follow(a[i][0]);
            }
        }
    }
}

```

Output:

```
mec@cl1-1-1:~/CS7B$ gcc pg2.c
mec@cl1-1-1:~/CS7B$ ./a.out
Enter the no of productions:
3
Enter the productions:
A=abc
A=def
A=ghi
Enter the elemets whose fisrt & follow is to be found:A
First(A)={adg}
Follow(A)={$}
Continue(0/1)?0
mec@cl1-1-1:~/CS7B$ 
mec@cl1-1-1:~/CS7B$ ./a.out
Enter the no of productions:
3
Enter the productions:
B->AB
A->a
B->b
Enter the elemets whose fisrt & follow is to be found:A
First(A)={>}
Follow(A)={>>}
Continue(0/1)?1
Enter the elemets whose fisrt & follow is to be found:B
First(B)={>>}
Follow(B)={$}
Continue(0/1)?
```


Question 3

Construct a recursive descent parser for an expression.

Program :

```
#include<stdio.h>

char input[100];
int i = 0;
int curr = 0;

int E();
int Z();

int main() {
    printf("Enter input:\n");
    scanf("%s", input);
    while(input[i] != '\0')
        i++;
    // i contains length of the input.

    int res = E();
    if(res == 1 && curr == i)
        printf("Input has been accepted.\n");
    else
        printf("Input has been rejected.\n");
}

int E() {
    int res;
    // E -> iZ
    if(input[curr] == 'i') {
        curr++;
        res = Z();
        if(res == 1)
            return 1;
        else
            return -1;
    }
}

int Z() {
    int res;
    // Z -> +iZ
    if(input[curr] == '+' && input[curr + 1] == 'i') {
        curr += 2;
        res = Z();
        if(res == 1)
            return 1;
    }
    //Z -> e
    //not incrementing curr
}
```

```
        return 1;  
    }
```

Output:

```
C:\Users\malav\OneDrive\Documents\CDLab\Cycle3\Pg3>a  
Enter input:  
i+i  
Input has been accepted.  
  
C:\Users\malav\OneDrive\Documents\CDLab\Cycle3\Pg3>a  
Enter input:  
i-i  
Input has been rejected.
```

Question 4

Construct a Shift Reduce Parser for a given language.

Program :

```
#include <stdio.h>

char input[100];
int len = -1;
char stack[100];
int top = -1;

void display(int idx, char* action) {
    // Stack
    for(int i = 0; i <= top; i++)
        printf("%c", stack[i]);
    printf("\t\t");
    // Input
    for(int i = idx; i < len; i++)
        printf("%c", input[i]);
    // Action
    printf("$\t\t%s\n", action);
}

void checkForReduce(int i) {
    int checkFurther = 1;
    while(checkFurther) {
        checkFurther = 0;
        // S->S+S
        if (stack[top-2] == 'S' && stack[top-1] == '+' && stack[top]
== 'S') {
            display(i, "REDUCE");
            stack[top-2] = 'S';
            top -= 2;
            checkFurther = 1;
        }
        // S->S-S
        if (stack[top-2] == 'S' && stack[top-1] == '-' && stack[top]
== 'S') {
            display(i, "REDUCE");
            stack[top-2] = 'S';
            top -= 2;
            checkFurther = 1;
        }
        // S->(S)
        if (stack[top-2] == '(' && stack[top-1] == 'S' && stack[top]
== ')') {
            display(i, "REDUCE");
            stack[top-2] = 'S';
            top -= 2;
            checkFurther = 1;
        }
    }
}
```

```

        // S->i
        if (stack[top] == 'i') {
            display(i, "REDUCE");
            stack[top] = 'S';
            checkFurther = 1;
        }
    }
}

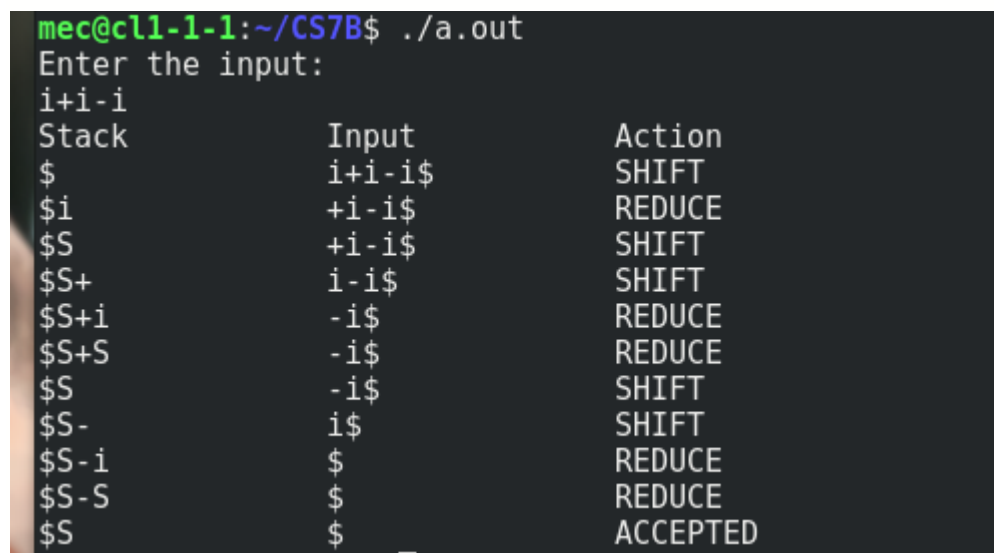
int main() {
    int i;
    printf("Enter the input:\n");
    scanf("%s", input);

    // calculate the input length in len.
    while(input[++len]!='\0');

    stack[++top] = '$';
    printf("Stack\t\tInput\t\tAction\n");
    for(i = 0; i < len; i++) {
        display(i, "SHIFT");
        stack[++top] = input[i];
        // parse from next input char i.e. i+1
        checkForReduce(i+1);
    }
    // Accepted.
    if(top == 1 && stack[top] == 'S')
        display(i, "ACCEPTED");
    // Rejected.
    else
        display(i, "REJECTED");
}

```

Output:



```

mec@cll-1-1:~/CS7B$ ./a.out
Enter the input:
i+i-i
Stack      Input      Action
$          i+i-i$    SHIFT
$i         +i-i$    REDUCE
$$         +i-i$    SHIFT
$$+        i-i$     SHIFT
$$+i       -i$     REDUCE
$$+S       -i$     REDUCE
$$S        -i$     SHIFT
$$S-       i$     SHIFT
$$S-i      $      REDUCE
$$S-S      $      REDUCE
$$S        $      ACCEPTED

```

Question 1

Implement Intermediate code generation for simple expressions.

Program :

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

void checkForOperator(char* input, char operator, char* reg) {
    int i = 0, j = 0;
    char temp[100];
    while(input[i] != '\0') {
        if(input[i] == operator) {
            printf("%c\t\t%c\t\t%c\t\t%c\n", operator, *reg,
input[i-1], input[i+1]);
            temp[j-1] = *reg;
            i += 2;
            (*reg)--;
            continue;
        }
        temp[j] = input[i];
        i++;
        j++;
    }
    temp[++j] = '\0';
    strcpy(input, temp);
}

void generateIntermediateCode(char* input) {
    char reg = 'Z';
    checkForOperator(input, '/', &reg);
    checkForOperator(input, '*', &reg);
    checkForOperator(input, '+', &reg);
    checkForOperator(input, '-', &reg);
    checkForOperator(input, '=', &reg);
}

void main() {
    char input[100];
    printf("Enter the expression: ");
    scanf("%s", input);
    printf("Operator\tDestination\tOperand1\tOperand2\n");
    generateIntermediateCode(input);
}
```

Output:

```
Enter the expression: a+b/g-f*r
Operator      Destination  Operand1  Operand2
/             Z          b         g
*             Y          f         r
+             X          a         Z
-             W          X         Y
```

Question 2

Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc

Program :

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

void main()
{
    char icode[10][30], str[20], opr[10];
    int i = 0;
    //clrscr();
    printf("\n Enter the set of intermediate code (terminated by
exit):\n");
    do
    {
        scanf("%s", icode[i]);
    } while (strcmp(icode[i++], "exit") != 0);
    printf("\n target code generation");
    printf("\n*****");
    i = 0;
    do {
        strcpy(str, icode[i]);
        switch (str[3]) {
            case '+':
                strcpy(opr, "ADD ");
                break;
            case '-':
                strcpy(opr, "SUB ");
                break;
            case '*':
                strcpy(opr, "MUL ");
                break;
            case '/':
                strcpy(opr, "DIV ");
                break;
        }
        printf("\n\tMov %c,R%d", str[2], i);
        printf("\n\t%s%c,R%d", opr, str[4], i);
        printf("\n\tMov R%d,%c", i, str[0]);
    } while (strcmp(icode[++i], "exit") != 0);
    //getch();
}
```

Output:

```
mec@cl1-1-1: ~/CS7B
File Edit View Search Terminal Help
mec@cl1-1-1:~/CS7B$ gcc pg4-2.c
mec@cl1-1-1:~/CS7B$ ./a.out

Enter the set of intermediate code (terminated by exit):
a=a+b
c=f/h
e=u+t
k=a*g
exit

target code generation
*****
      Mov a,R0
      ADD b,R0
      Mov R0,a
      Mov f,R1
      DIV h,R1
      Mov R1,c
      Mov u,R2
      ADD t,R2
      Mov R2,e
      Mov a,R3
      MUL g,R3
      Mov R3,kmec@cl1-1-1:~/CS7B$
```