Formal Language and Automata Theory Record

Submitted in partial fulfilment of the requirements for Degree
Of
Bachelor of Technology
In
Computer Science & Engineering

Submitted
By

Name: - Akash Kumar
(Regd. No. –1801289021)
Sec - A Group - 01 Roll no. - 06
Branch - CSE
Semester - 5th

To

TRIDENT ACADEMY OF TECHNOLOGY

BHUBANESWAR

Session - 2020 - 21

Vision & Mission of Dept. Of CSE

Vision

Our aim is to build computer science professionals with strong leadership quality, discipline and passion filled ability to encounter global challenges with technological insight with reference to current context.

Mission

To build the Department of CSE, oriented in a direction that ensures:

1. To produce Quality Computer Engineers, IT Professionals and Entrepreneurs capable of meeting the need of the Nation with innovativeness,
2. To form educational pathway geared up to produce competent Computer Professionals capable of maximizing their career choice and options at par and in pace with Global Standard and evolving technologies, and
3. To contribute to the society with honesty and integrity through innovative research in the multi-disciplinary areas of evolving and upcoming technologies".

TRIDENT ACADEMY OF TECHNOLOGY, BHUBANESWAR

DEPARTMENT OF CSE / IT

Subject:

Name: ……..Akash Kumar………………………….. Branch......CSE...............Section....A..

Regd. No.........1801289021............... Roll No....06...Sign...

Lab No.	Lab Date	Assignments	Pg. No	Submission Date
1		1.DFA String starting with 'a' 2. DFA string ending with 'a' 3.DFA Ending with 'aa'	04 -07	
2		1.DFA string accepting starting with a and ending with a. 2. DFA accept the string "**nano**". 3.FA of Regular Expression (a+aa*b) *.	08- 11	
3		1.implementation of DFA that accepts all even integers		
		2.implementation of DFA that accepts all odd integers	12-15	
4		1. Programme for Implementation NFA to DFA Conversion	16-19	
5		1.Programme for implementation ϵ-NFA to DFA conversion	20-28	
6		1.Programme for the implementation for DFA Minimization.		
		2.Quesion Using JFLAB	29-39	
7		1.Programme for design and implementation of PDA. 2. Questions Using JFLAB	40-43	
8		1.CYK parsing algorithm for some specific GFG using JFLAB	44	
9		1.Turing machine for some Recursively Languages using JFLAB.	45-47	

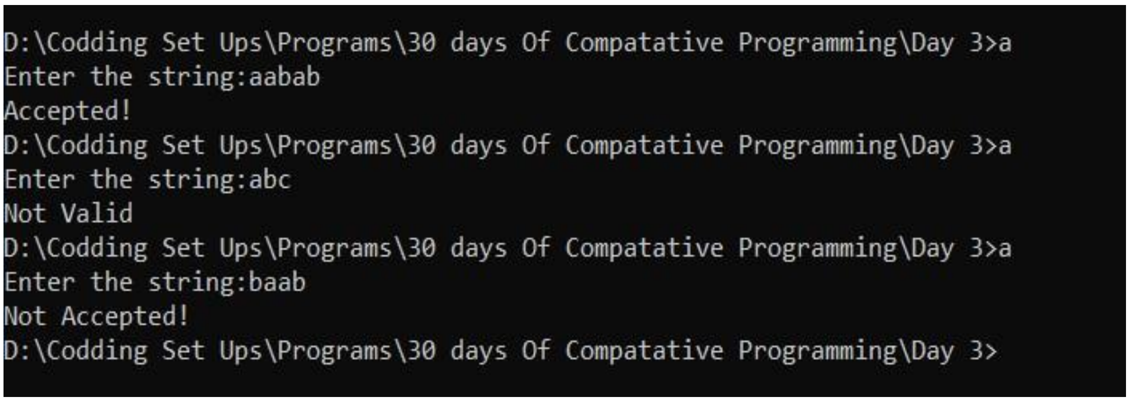
<u>ASSIGNMENT-1</u>

Q1.WAP in C to construct a DFA over input alphabets {a,b} accepting all strings starting with ‘a’

Code:

```
/*Akash Kumar
  1801289021
*/
#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='a' && s[i]!='b')
        return 0;

}
return 1;




}
 main(){
    char s[20];
    printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return;
    }

    if(s[0]=='a')
        printf("Accepted!");
    else
        printf("Not Accepted!");
}
```

Output



```
D:\Codding Set Ups\Programs\30 days Of Compatative Programming\Day 3>a
Enter the string:aabab
Accepted!
D:\Codding Set Ups\Programs\30 days Of Compatative Programming\Day 3>a
Enter the string:abc
Not Valid
D:\Codding Set Ups\Programs\30 days Of Compatative Programming\Day 3>a
Enter the string:baab
Not Accepted!
D:\Codding Set Ups\Programs\30 days Of Compatative Programming\Day 3>
```

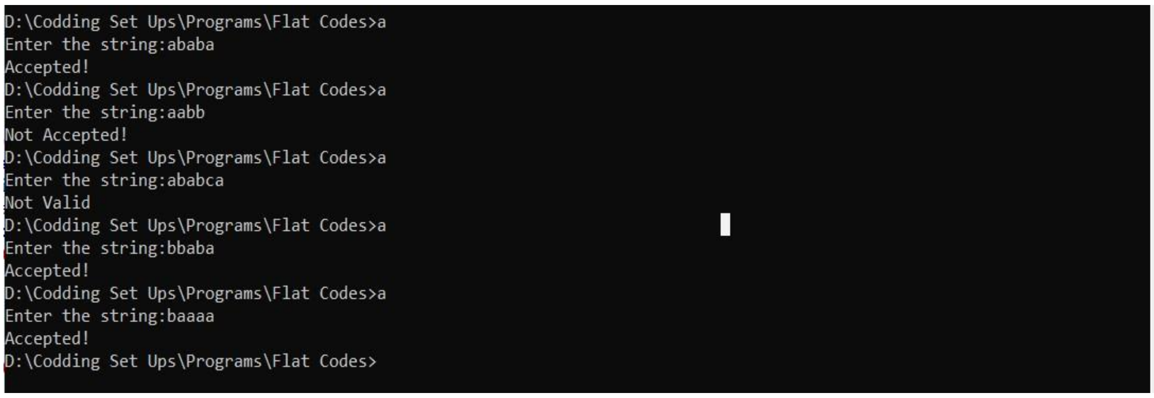
Q2.WAP in C construct a DFA over input alphabets {a,b} accepting all strings ending with 'a'.

Code:

```
/*Akash Kumar
  1801289021
*/
#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='a' && s[i]!='b')
        return 0;

}
return 1;
}
 main(){
    char s[20];
    printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return;
    }   if(s[n-1]=='a')
        printf("Accepted!");
```

Output

```
                      Accepted!");
}
```



```
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ababa
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aabb
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ababca
Not Valid
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:bbaba
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:baaaa
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>
```

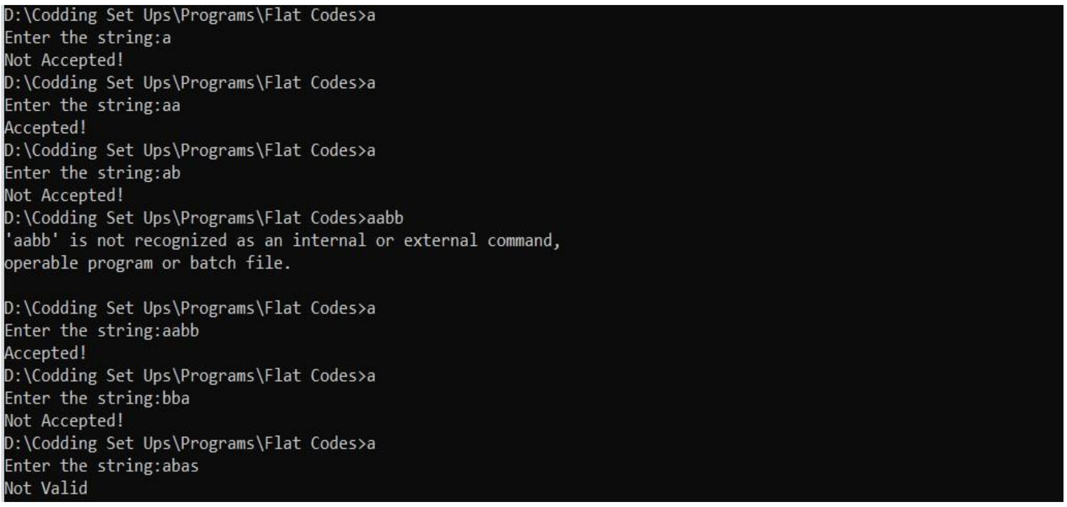
Q3.WAP in C to construct a DFA over input alphabets {a,b} accepting all strings starting

with ‘aa’.

Code:

```
/*Akash Kumar
  1801289021
*/
#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='a' && s[i]!='b')
        return 0;

}
return 1;

}
 main(){
    char s[20];
    printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return;
    }

    if(s[0]=='a' && s[1]=='a')
        printf("Accepted!");
    else
        printf("Not Accepted!");
}
```

Output:



```
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:a
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aa
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ab
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>aabb
'aabb' is not recognized as an internal or external command,
operable program or batch file.

D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aabb
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:bba
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:abas
Not Valid
```

Name - Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A

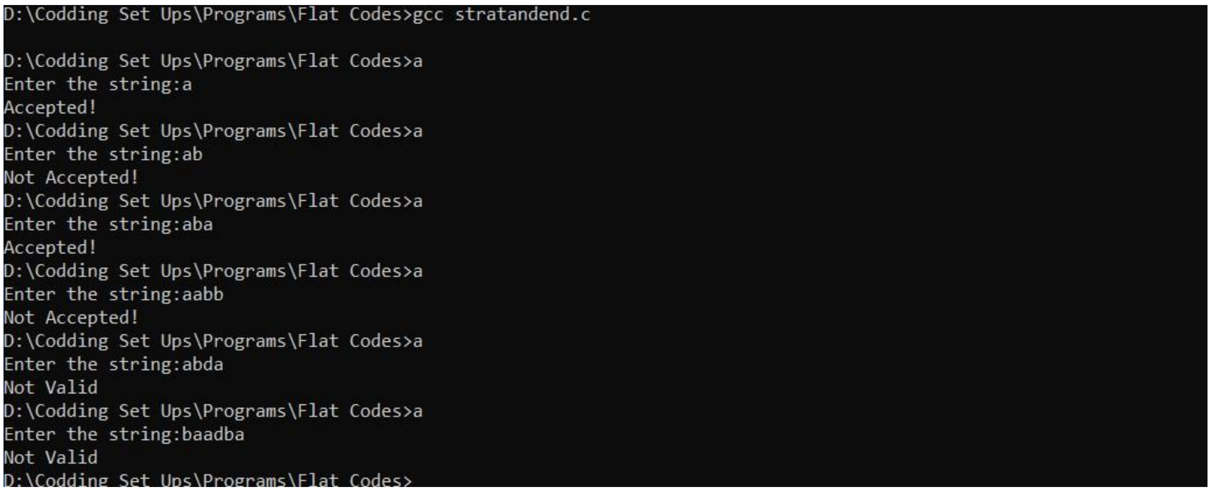
ASSIGNMENT-2

Q1. WAP in C to construct a DFA over input alphabets {a,b} accepting all strings starting with 'a' and ending with 'a'.

Code:

```
/*Akash Kumar
  1801289021
*/
#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='a' && s[i]!='b')
        return 0;

}
return 1;

}
 main(){
    char s[20];
    printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return;
    }

    if(s[0]=='a' && s[n-1]=='a')
        printf("Accepted!");
    else
        printf("Not Accepted!");
}
```

Output



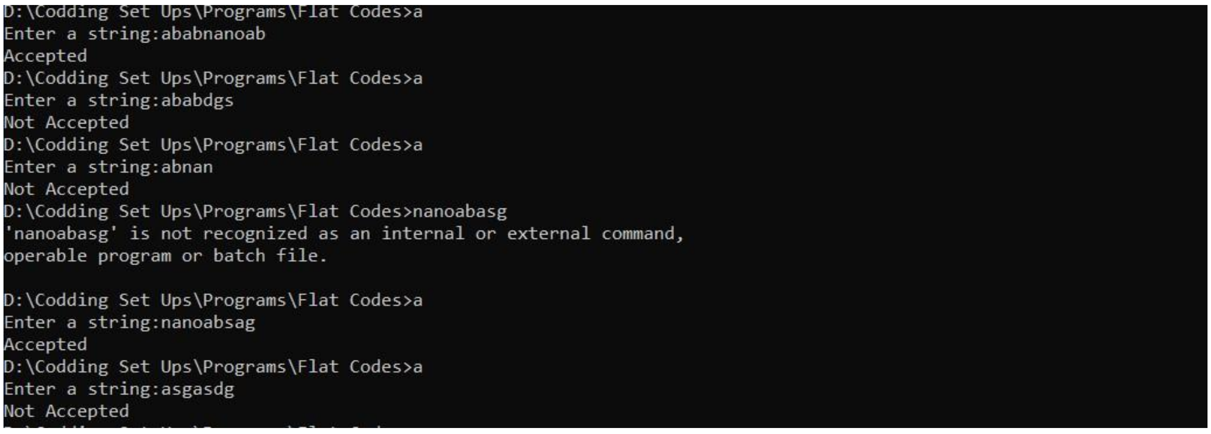
```
D:\Codding Set Ups\Programs\Flat Codes>gcc stratandend.c

D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:a
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ab
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aba
Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aabb
Not Accepted!
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:abda
Not Valid
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:baadba
Not Valid
D:\Codding Set Ups\Programs\Flat Codes>
```

Q2.WAP in C to accept the string “nano” and will reject all other strings.

Code:

```
/*Akash Kumar 1801289021 */
#include<stdio.h>

int main()
{
    char str[80], search[] = "nano";
    int count1 = 0, count2 = 0, i, j, flag;

    printf("Enter a string:");
    gets(str);
while (str[count1] != '\0')
        count1++;
    while (search[count2] != '\0')
        count2++;
    for (i = 0; i <= count1 - count2; i++)
    {
        for (j = i; j < i + count2; j++)
        {
            flag = 1;
            if (str[j] != search[j - i])
            {
                flag = 0;
                break;
            }
        }
        if (flag == 1)
            break;
    }
    if (flag == 1)
        printf("Accepted");
    else
        printf("Not Accepted");    return 0;
}
```

Output:



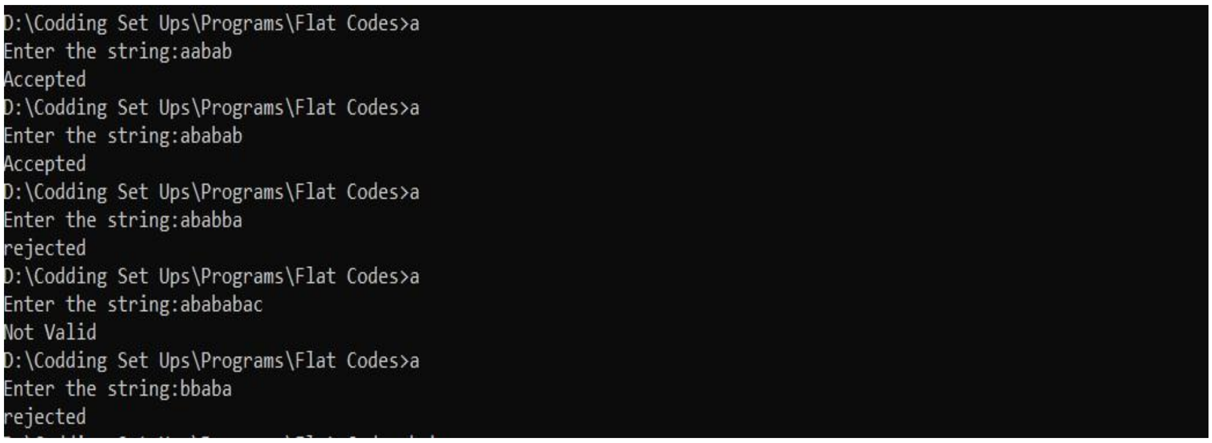
```
D:\Codding Set Ups\Programs\Flat Codes>a
Enter a string:ababnanoab
Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter a string:ababdgs
Not Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter a string:abnan
Not Accepted
D:\Codding Set Ups\Programs\Flat Codes>nanoabasg
'nanoabasg' is not recognized as an internal or external command,
operable program or batch file.

D:\Codding Set Ups\Programs\Flat Codes>a
Enter a string:nanoabsag
Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter a string:asgasdg
Not Accepted
```

Q3.WAP in C for implementing DFA of Regular Expression(a+aa*b) *. This string can only start by 'a' and no two 'b' comes together.

Code:

```
/*Akash Kumar 1801289021 */
#include<stdio.h>

#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='a' && s[i]!='b')
        return 0;
}
return 1;}
int  main(){
    char s[20];
printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return 0;
    }
if(s[0]=='a'){
        for(int i=1;i<n-1;i++){
            if(s[i] == 'b'){

                if(s[i+1]=='b')
                        printf("rejected");
                return 0;           }
            }
        }
   printf("Accepted");
}else{
             printf("rejected");}
}
```

Output


```
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:aabab
Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ababab
Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:ababba
rejected
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:abababac
Not Valid
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:bbaba
rejected
```

Name - Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A

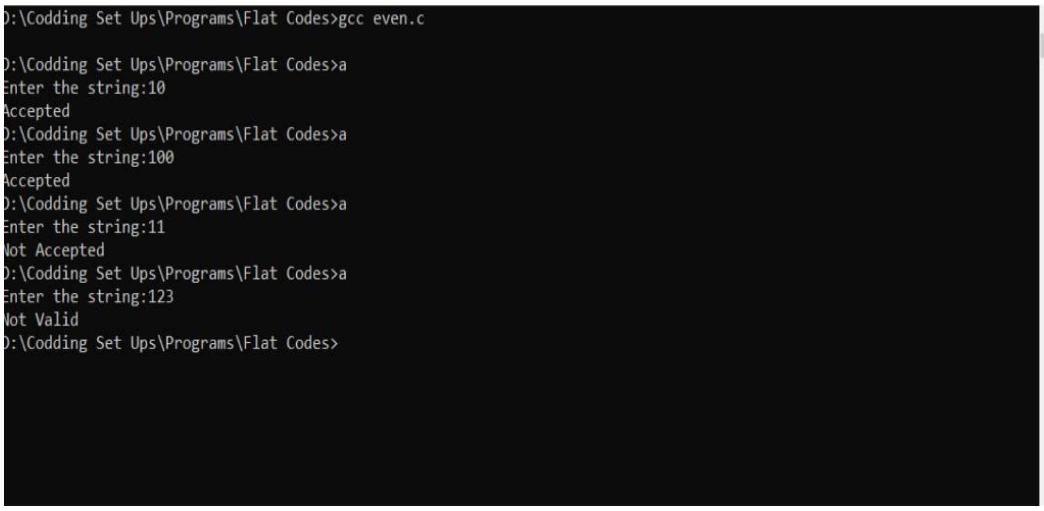
ASSIGNMENT-3

Q1.WAP in C for the implementation of DFA that accepts all even integers only in the form of Binary strings.

Code:

```
/*Akash Kumar 1801289021 */
#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='0' && s[i]!='1')
        return 0;}
return 1;
}
 int main(){
    char s[10];
    int i=0;
    char state='0';
   printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return 0;
    }

    while(s[i]!='\0'){
        switch(s[i]){

        case '0':state='1';i++;
                 break;
        case '1': state='0';i++;
        break;
        default:
            printf("NOt Valid");
}
    }
if(state=='1'){
        printf("Accepted");
        return 0;
}else{
        printf("Not Accepted");
        return 0;
    }
    return 0;

}
```

Code:



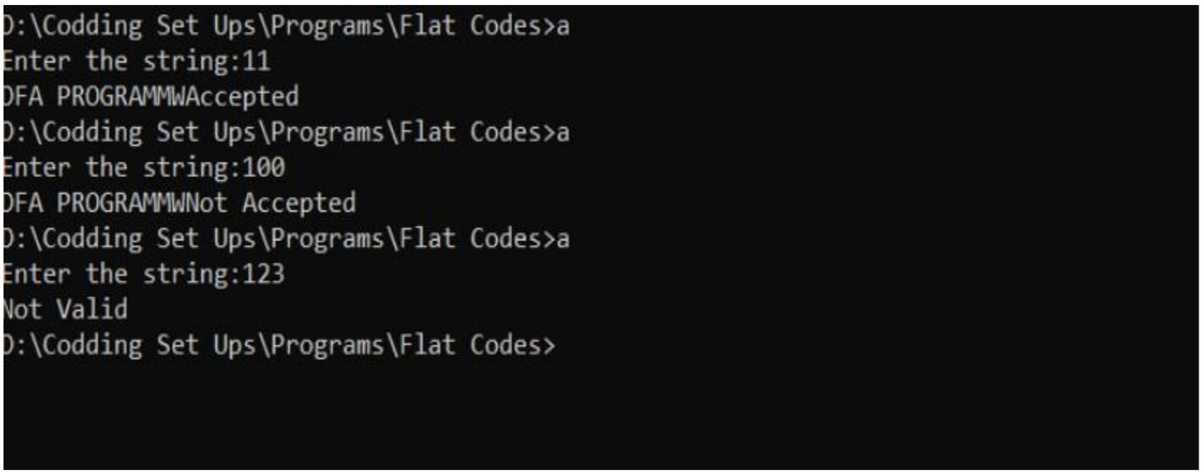
```
):\Codding Set Ups\Programs\Flat Codes>gcc even.c

):\Codding Set Ups\Programs\Flat Codes>a
Enter the string:10
Accepted
):\Codding Set Ups\Programs\Flat Codes>a
Enter the string:100
Accepted
):\Codding Set Ups\Programs\Flat Codes>a
Enter the string:11
Not Accepted
):\Codding Set Ups\Programs\Flat Codes>a
Enter the string:123
Not Valid
):\Codding Set Ups\Programs\Flat Codes>
```

Q2.WAP in C for Implementation of DFA that accepts all odd integers only in the form of binary strings.

Code:

```
/*Akash Kumar 1801289021 */

#include<stdio.h>
#include<string.h>
int validation(char s[],int n){
for(int i=0;i<n;i++){
    if(s[i]!='0' && s[i]!='1')
        return 0;}
return 1;
}
 int main(){
    char s[10];
    int i=0;
    char state='0';
   printf("Enter the string:");
    scanf("%s",&s);
    int n=strlen(s);
    if(validation(s,n)==0){
        printf("Not Valid");
        return 0;
    }

    while(s[i]!='\0'){
        switch(s[i]){

        case '0':state='0';i++;
                 break;
        case '1': state='1';i++;
        break;
        default:
            printf("NOt Valid");
}
    }
    printf("DFA PROGRAMME");
if(state=='1'){
        printf("Accepted");
        return 0;
}else{
        printf("Not Accepted");
        return 0;
    }
    return 0;

}
```

Output



```
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:11
DFA PROGRAMMWAccepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:100
DFA PROGRAMMWNot Accepted
D:\Codding Set Ups\Programs\Flat Codes>a
Enter the string:123
Not Valid
D:\Codding Set Ups\Programs\Flat Codes>
```

Name - Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A

ASSIGNMENT-4

Q1.WAP for the implementation NFA to DFA Conversion.

Code

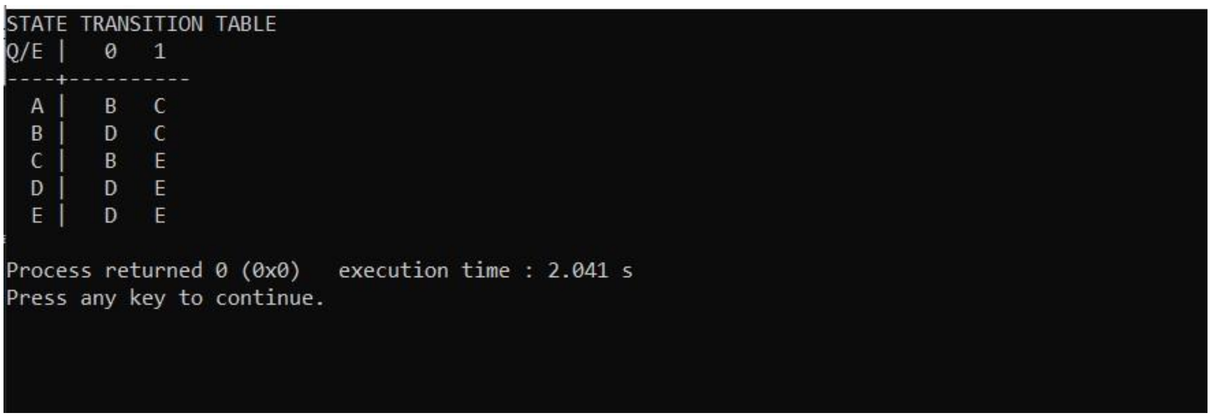
```
/*Akash Kumar 1801289021 */
#include<stdio.h>
#include<string.h>
#define STATES 256
#define SYMBOLS 20
int N_symbols;
int NFA_states;
char *NFAtab[STATES][SYMBOLS];
int DFA_states;
int DFAtab[STATES][SYMBOLS];
void put_dfa_table( int tab[][SYMBOLS],
   int nstates,
   int nsymbols){
int i, j;
puts("STATE TRANSITION TABLE");
printf ("Q/E");
printf (" |");
for (i = 0; i <nsymbols; i++)
printf(" %3c",'0'+i);
printf ("\n----+--");
for (i = 0; i <nsymbols; i++)
printf ("----");
printf ("\n");
for (i = 0; i <nstates; i++)
 {
printf(" %2c | ",'A'+i);
for (j= 0; j <nsymbols;j++)
printf(" %2c ",'A'+tab[i][j]);
printf("\n");
 }


}
void init_NFA_table()
```

```
{
NFAtab[0][0] ="12";
NFAtab[0][1] ="13";
NFAtab[1][0] ="12";
NFAtab[1][1] ="13";
NFAtab[2][0] ="4";
NFAtab[2][1] ="";
NFAtab[3][0] ="";
NFAtab[3][1] ="4";
NFAtab[4][0] ="4";
NFAtab[4][1] ="4";
NFA_states = 5;
DFA_states = 0;
N_symbols = 2;
}
void string_merge(char *s, char *t)
{
char temp[STATES], *r=temp, *p=s;

while (*p && *t)
   {
if(*p ==*t)
       {
          *r++ = *p++;
t++;
       }

else if(*p < *t)
   {
         *r++ = *p++;
   }
else    {
         *r++ = *t++;
   }
}



*r='\0';
if(*p)strcat(r,p);
```

```
else if(*t)strcat(r,t);
strcpy(s,temp);
}

void get_next_state(char *nextstates, char *cur_states, char *nfa[STATES][SYMBOLS] ,int n_nfa, int symbol)
 {



int i;
char temp[STATES];

temp[0] = '\0';
for (i = 0; i <strlen(cur_states); i++)
string_merge(temp, nfa[cur_states[i]-'0'][symbol]);
strcpy(nextstates, temp);
 }

int state_index(char *state, char statename[][STATES], int *pn)
{



int i;
if (!*state)
return -1 ;

for (i = 0; i < *pn; i++)
if (!strcmp(state, statename[i]))
return i;
strcpy(statename[i], state);
return(*pn)++;
}int nfa_to_dfa(char *nfa[STATES][SYMBOLS],int n_nfa,int n_sym, int dfa[][SYMBOLS])
{
char statename[STATES][STATES];
int i = 0;
int n = 1;
char nextstate[STATES];
int j;
strcpy(statename[0], "0");
```

```
for (i = 0; i <n;i++)
{
for (j= 0; j <n_sym; j++)
{
get_next_state(nextstate, statename[i], nfa, n_nfa, j);
dfa[i][j]= state_index(nextstate, statename, &n);
}
}
return n;
}
int main()
{
init_NFA_table();
DFA_states = nfa_to_dfa(NFAtab, NFA_states, N_symbols, DFAtab);
put_dfa_table(DFAtab, DFA_states, N_symbols);
return 0;

}
```

Output:



```
STATE TRANSITION TABLE
Q/E |   0   1
----+----------
  A |   B   C
  B |   D   C
  C |   B   E
  D |   D   E
  E |   D   E

Process returned 0 (0x0)   execution time : 2.041 s
Press any key to continue.
```

Name - Akash Kumar
RegNo -1801289021
RollNo-06
Branch-CSE-A

ASSIGNMENT-5

Q1.WAP for the implementation ϵ-NFA to DFA conversion.

Code:

```
/*Akash Kumar 1801289021 */
#include <stdio.h>

#include <string.h>


#define STATES 99

#define SYMBOLS 20


int N_symbols;

int N_NFA_states;

char *NFAtab[STATES][SYMBOLS];

char *NFA_finals;


int N_DFA_states;

int DFAtab[STATES][SYMBOLS];

char DFA_finals[STATES+1];


char StateName[STATES][STATES+1];

char Eclosure[STATES][STATES+1];



void print_nfa_table(

            char *tab[][SYMBOLS],

            int nstates,

            char *finals)

{

            int i, j;


            puts("\nNFA: STATE TRANSITION TABLE");



            printf("     | ");

            for (i = 0; i <nsymbols; i++) printf("  %-6c", '0'+i);

            printf("  e\n");
```

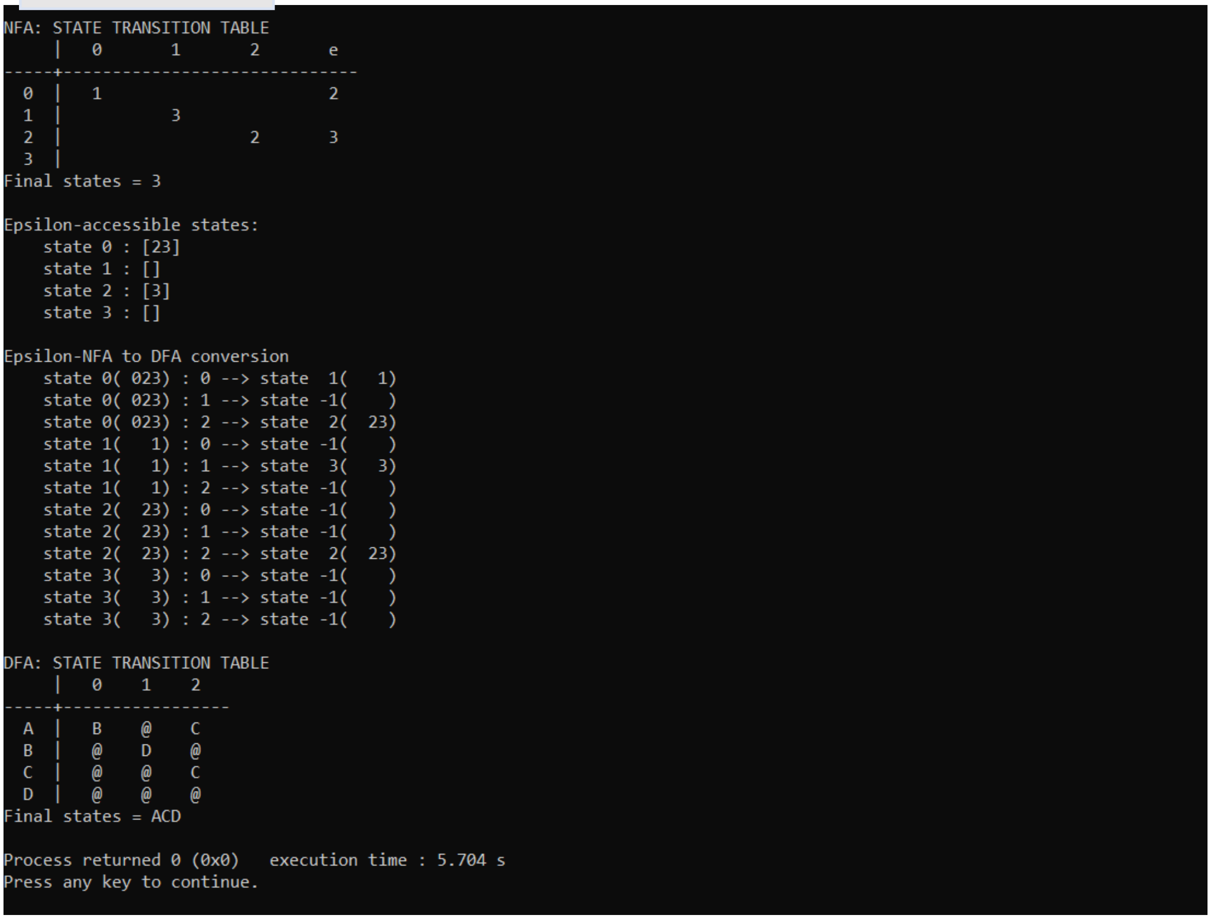
```
        printf("-----+--");
        for (i = 0; i < nsymbols+1; i++) printf("-------");
        printf("\n");

        for (i = 0; i <nstates; i++)
    {
                printf("  %c  | ", '0'+i);
                for (j = 0; j < nsymbols+1; j++)
                        printf("  %-6s", tab[i][j]);
                printf("\n");
        }
        printf("Final states = %s\n", finals);
}



void print_dfa_table(
        int tab[][SYMBOLS],
        int nstates,
        int nsymbols,
        char *finals)
{
        int i, j;

        puts("\nDFA: STATE TRANSITION TABLE");



        printf("     | ");
        for (i = 0; i <nsymbols; i++)
    printf("  %c  ", '0'+i);

        printf("\n-----+--");
        for (i = 0; i <nsymbols; i++)
    printf("-----");
        printf("\n");

        for (i = 0; i <nstates; i++)
    {
                printf("  %c  | ", 'A'+i);
                for (j = 0; j <nsymbols; j++)
```

```
                                printf("  %c  ", tab[i][j]);
                        printf("\n");
        }
                printf("Final states = %s\n", finals);
}



void load_NFA_table()
{

        NFAtab[0][0] = "1";
        NFAtab[0][1] = "";
        NFAtab[0][2] = "";
        NFAtab[0][3] = "2";
        NFAtab[1][0] = "";
        NFAtab[1][1] = "3";
        NFAtab[1][2] = "";
        NFAtab[1][3] = "";
        NFAtab[2][0] = "";
        NFAtab[2][1] = "";
        NFAtab[2][2] = "2";
        NFAtab[2][3] = "3";
        NFAtab[3][0] = "";
        NFAtab[3][1] = "";
        NFAtab[3][2] = "";
        NFAtab[3][3] = "";

        N_symbols = 3;
        N_NFA_states = 4;
        NFA_finals = "3";
        N_DFA_states = 0;
}

int string_merge(char *s, char *t)
{
        int n=0;
        char temp[STATES+1], *r=temp, *p=s;

        while (*p && *t)
        {
                        if (*p == *t)
                        {
                                *r++ = *p++;
            t++;
                        }
else if (*p < *t)
        {
                                *r++ = *p++;
                        }
else
        {
                                *r++ = *t++;
                                n++;
                        }
                }
                *r = '\0';

                if (*t)
        {
                        strcat(r, t);
                        n += strlen(t);
                  }
else if (*p)
strcat(r, p);

                strcpy(s, temp);

                return n;
}

void get_next_state_NFA(char *nextstates, char *cur_states,
                char *nfa[STATES][SYMBOLS], int symbol)
{
                int i;
                char temp[STATES+1];

                temp[0] = '\0';
                for (i = 0; i <strlen(cur_states); i++)
```

```
                        string_merge(temp, nfa[cur_states[i]-'0'][symbol]);
        strcpy(nextstates, temp);
}


int state_index(char *state, char stnt[][STATES+1], int *pn)
{
        int i;

        if (!*state)
   return -1;

        for (i = 0; i < *pn; i++)
                if (!strcmp(state, stnt[i]))
     return i;

        strcpy(stnt[i], state);
        return (*pn)++;
}



void get_ep_states(int state, char *epstates,
        char *nfa[][SYMBOLS], int n_sym)
{
        int i, n=0;


        strcpy(epstates, nfa[state][n_sym]);

        do {
                for (i = 0; i <strlen(epstates); i++)
                        n = string_merge(epstates, nfa[epstates[i]-'0'][n_sym]);
        } while (n);
}

void init_Eclosure(char eclosure[][STATES+1],
        char *nfa[][SYMBOLS], int n_nfa, int n_sym)
{
        int i;
        printf("\nEpsilon-accessible states:\n");
        for (i = 0; i <n_nfa; i++)
    {
                get_ep_states(i, eclosure[i], nfa, n_sym);
                printf("    state %d : [%s]\n", i, eclosure[i]);
        }
        printf("\n");
}



void e_closure(char *epstates, char *states, char eclosure[][STATES+1])
{
        int i;

        strcpy(epstates, states);
        for (i = 0; i <strlen(states); i++)
                string_merge(epstates, eclosure[states[i]-'0']);
}



int nfa_to_dfa(char *nfa[][SYMBOLS], int n_nfa,
        int n_sym, int dfa[][SYMBOLS])
{
        int i = 0;
        int n = 1;

        char nextstate[STATES+1];
        char temp[STATES+1];
        int j;

        init_Eclosure(Eclosure, nfa, n_nfa, n_sym);

        e_closure(temp, "0", Eclosure);
        strcpy(StateName[0], temp);

        printf("Epsilon-NFA to DFA conversion\n");
        for (i = 0; i < n; i++) {
                for (j = 0; j <n_sym; j++) {
                        get_next_state_NFA(nextstate, StateName[i], nfa, j);
                        e_closure(temp, nextstate, Eclosure);
                        dfa[i][j] = state_index(temp, StateName, &n);
                        printf("    state %d(%4s) : %d --> state %2d(%4s)\n",
                                i, StateName[i], j, dfa[i][j], temp);
                        dfa[i][j] += 'A';
                }
        }

        return n;
}



void get_DFA_finals(
        char *dfinals,
        char *nfinals,
        char stnt[][STATES+1],
        int n_dfa)
{
        int i, j, k=0, n=strlen(nfinals);

        for (i = 0; i <n_dfa; i++)
    {
                for (j = 0; j < n; j++)
                 {
                        if (strchr(stnt[i], nfinals[j]))
                        {
                                dfinals[k++] = i+'A';
                                break;
                        }
                }
        }
        dfinals[k] = '\0';
}
int main()
{
        load_NFA_table();
        print_nfa_table(NFAtab, N_NFA_states, N_symbols, NFA_finals);
```

```
	N_DFA_states = nfa_to_dfa(NFAtab, N_NFA_states, N_symbols, DFAtab);
	get_DFA_finals(DFA_finals, NFA_finals, StateName, N_DFA_states);

	print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);
	return 0;
}
```

Output



```
NFA: STATE TRANSITION TABLE
     |   0       1       2       e
-----+----------------------------
  0  |   1                       2
  1  |           3
  2  |                   2       3
  3  |
Final states = 3

Epsilon-accessible states:
    state 0 : [23]
    state 1 : []
    state 2 : [3]
    state 3 : []

Epsilon-NFA to DFA conversion
    state 0( 023) : 0 --> state  1(   1)
    state 0( 023) : 1 --> state -1(    )
    state 0( 023) : 2 --> state  2(  23)
    state 1(   1) : 0 --> state -1(    )
    state 1(   1) : 1 --> state  3(   3)
    state 1(   1) : 2 --> state -1(    )
    state 2(  23) : 0 --> state -1(    )
    state 2(  23) : 1 --> state -1(    )
    state 2(  23) : 2 --> state  2(  23)
    state 3(   3) : 0 --> state -1(    )
    state 3(   3) : 1 --> state -1(    )
    state 3(   3) : 2 --> state -1(    )

DFA: STATE TRANSITION TABLE
     |   0    1    2
-----+----------------
  A  |   B    @    C
  B  |   @    D    @
  C  |   @    @    C
  D  |   @    @    @
Final states = ACD

Process returned 0 (0x0)   execution time : 5.704 s
Press any key to continue.
```

Name - Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A

ASSIGNMENT-6

Q1.WAP for the implementation for DFA Minimization.

Code:

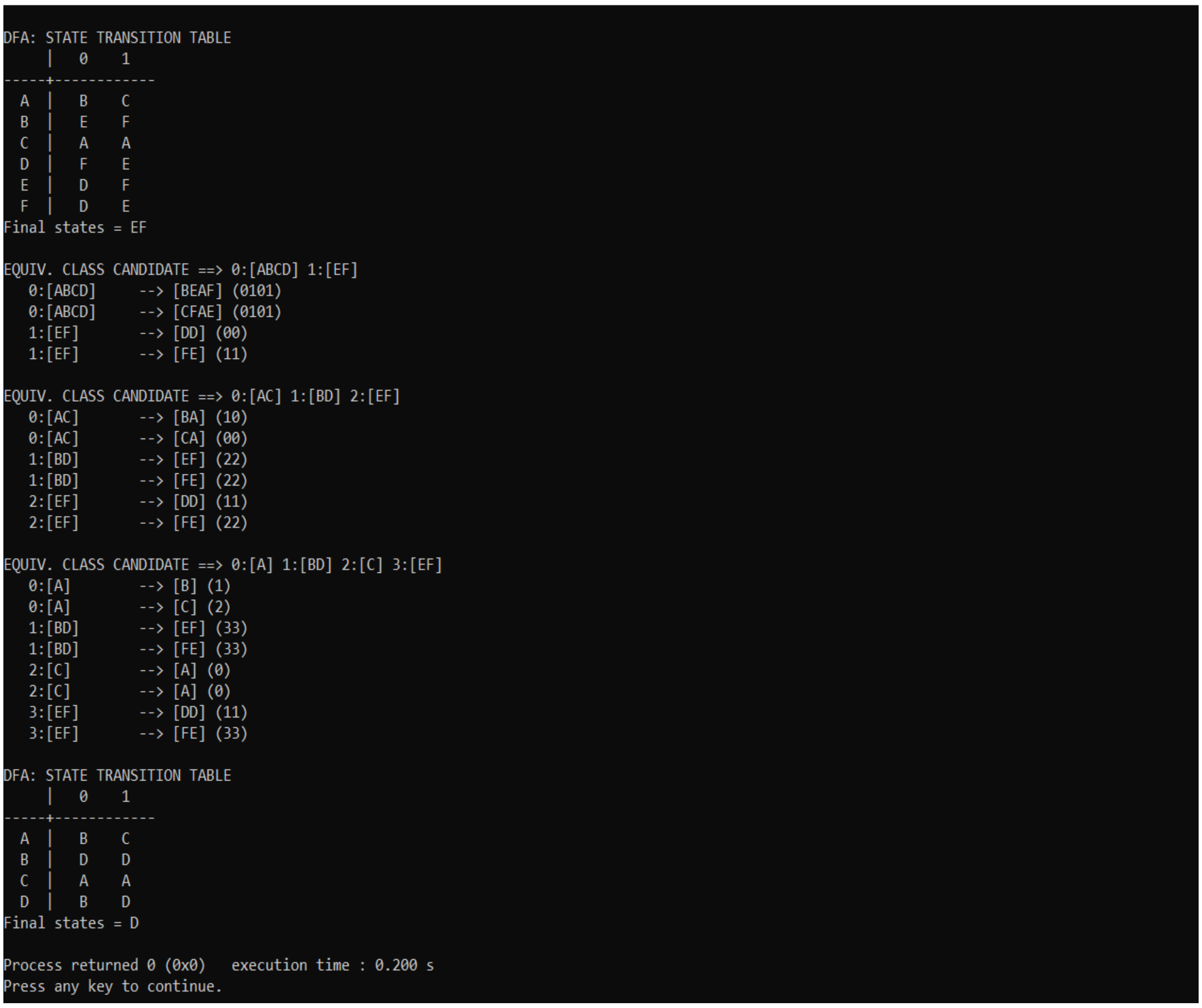
```
#include <stdio.h>
#include <string.h>
#define STATES          99
#define SYMBOLS  20
int N_symbols;
int N_DFA_states;
char *DFA_finals;
int DFAtab[STATES][SYMBOLS];

char StateName[STATES][STATES+1];

int N_optDFA_states;
int OptDFA[STATES][SYMBOLS];
char NEW_finals[STATES+1];



void print_dfa_table(
        int tab[][SYMBOLS],
        int nstates,
        int nsymbols,
        char *finals)
{
        int i, j;

        puts("\nDFA: STATE TRANSITION TABLE");
        printf("     | ");
        for (i = 0; i <nsymbols; i++)
   printf("  %c  ", '0'+i);

        printf("\n-----+--");
        for (i = 0; i <nsymbols; i++)
   printf("-----");
        printf("\n");
        for (i = 0; i <nstates; i++)
    {
                printf("  %c  | ", 'A'+i);
                for (j = 0; j <nsymbols; j++)
                        printf("  %c  ", tab[i][j]);
                printf("\n");
        }
        printf("Final states = %s\n", finals);
}



void load_DFA_table()
{

        DFAtab[0][0] = 'B'; DFAtab[0][1] = 'C';
        DFAtab[1][0] = 'E'; DFAtab[1][1] = 'F';
        DFAtab[2][0] = 'A'; DFAtab[2][1] = 'A';
        DFAtab[3][0] = 'F'; DFAtab[3][1] = 'E';
        DFAtab[4][0] = 'D'; DFAtab[4][1] = 'F';
        DFAtab[5][0] = 'D'; DFAtab[5][1] = 'E';

        DFA_finals = "EF";
        N_DFA_states = 6;
        N_symbols = 2;
}

void get_next_state(char *nextstates, char *cur_states,
                int dfa[STATES][SYMBOLS], int symbol){
        int i, ch;

        for (i = 0; i <strlen(cur_states); i++)
                *nextstates++ = dfa[cur_states[i]-'A'][symbol];
          *nextstates = '\0';
}



char equiv_class_ndx(char ch, char stnt[][STATES+1], int n)
{
```

```
            int i;

            for (i = 0; i < n; i++)
                        if (strchr(stnt[i], ch))
         return i+'0';
             return -1;
}

char is_one_nextstate(char *s)
{
            char equiv_class;

            while (*s == '@') s++;
            equiv_class = *s++;

            while (*s)
      {
                        if (*s != '@' && *s != equiv_class)
          return 0;
                            s++;
      }

            return equiv_class;
}

int state_index(char *state, char stnt[][STATES+1], int n, int *pn,
            int cur)
{
            int i;
            char state_flags[STATES+1];

            if (!*state)
      return -1;

            for (i = 0; i <strlen(state); i++)
                        state_flags[i] = equiv_class_ndx(state[i], stnt, n);
               state_flags[i] = '\0';

                printf("   %d:[%s]\t--> [%s] (%s)\n",
                cur, stnt[cur], state, state_flags);

        if (i=is_one_nextstate(state_flags))
                return i-'0';
        else
      {
                 strcpy(stnt[*pn], state_flags);
                 return (*pn)++;
            }
}

int init_equiv_class(char statename[][STATES+1], int n, char *finals)
{
        int i, j;

        if (strlen(finals) == n) {
                strcpy(statename[0], finals);
                return 1;
        }

        strcpy(statename[1], finals);

        for (i=j=0; i < n; i++) {
                if (i == *finals-'A') {
                        finals++;
                } else statename[0][j++] = i+'A';
        }
        statename[0][j] = '\0';

        return 2;
}

int get_optimized_DFA(char stnt[][STATES+1], int n,
        int dfa[][SYMBOLS], int n_sym, int newdfa[][SYMBOLS])
{
        int n2=n;
        int i, j;
        char nextstate[STATES+1];
        for (i = 0; i < n; i++) {
                for (j = 0; j <n_sym; j++) {
                        get_next_state(nextstate, stnt[i], dfa, j);
                        newdfa[i][j] = state_index(nextstate, stnt, n, &n2, i)+'A';
                }
        }

        return n2;
}



void chr_append(char *s, char ch)
{
        int n=strlen(s);

        *(s+n) = ch;
        *(s+n+1) = '\0';
}

void sort(char stnt[][STATES+1], int n)
{
        int i, j;
        char temp[STATES+1];

        for (i = 0; i < n-1; i++)
                for (j = i+1; j < n; j++)
                        if (stnt[i][0] >stnt[j][0])
          {
                                strcpy(temp, stnt[i]);
                                strcpy(stnt[i], stnt[j]);
                                strcpy(stnt[j], temp);
                          }
}

int split_equiv_class(char stnt[][STATES+1],
        int i1,
        int i2,
        int n,
        int n_dfa)
```

```
{
        char *old=stnt[i1], *vec=stnt[i2];
        int i, n2, flag=0;
        char newstates[STATES][STATES+1];

        for (i=0; i < STATES; i++) newstates[i][0] = '\0';

        for (i=0; vec[i]; i++)
                chr_append(newstates[vec[i]-'0'], old[i]);

        for (i=0, n2=n; i <n_dfa; i++) {
                if (newstates[i][0]) {
                        if (!flag) {
                                strcpy(stnt[i1], newstates[i]);
                                flag = 1;
                        } else
                                strcpy(stnt[n2++], newstates[i]);
                }
        }

        sort(stnt, n2);

        return n2;
}



int set_new_equiv_class(char stnt[][STATES+1], int n,
        int newdfa[][SYMBOLS], int n_sym, int n_dfa)
{
        int i, j, k;

        for (i = 0; i < n; i++) {
                for (j = 0; j <n_sym; j++) {
                        k = newdfa[i][j]-'A';
                        if (k >= n)
                                return split_equiv_class(stnt, i, k, n, n_dfa);
                }
        }
        return n;
}

void print_equiv_classes(char stnt[][STATES+1], int n)
{
        int i;

        printf("\nEQUIV. CLASS CANDIDATE ==>");
        for (i = 0; i < n; i++)
                printf(" %d:[%s]", i, stnt[i]);
        printf("\n");
}



int optimize_DFA(
        int dfa[][SYMBOLS],
        int n_dfa,
        int n_sym,
        char *finals,
        char stnt[][STATES+1],
        int newdfa[][SYMBOLS])
{
        char nextstate[STATES+1];
        int n;
        int n2;
        n = init_equiv_class(stnt, n_dfa, finals);

        while (1) {
                print_equiv_classes(stnt, n);
                n2 = get_optimized_DFA(stnt, n, dfa, n_sym, newdfa);
                if (n != n2)
                        n = set_new_equiv_class(stnt, n, newdfa, n_sym, n_dfa);
                else
break;
        }

        return n;
}
```

```
int is_subset(char *s, char *t)
{
        int i;

        for (i = 0; *t; i++)
                if (!strchr(s, *t++))
       return 0;
          return 1;
}

void get_NEW_finals(
        char *newfinals,
        char *oldfinals,
        char stnt[][STATES+1],
        int n)
{
        int i;

        for (i = 0; i < n; i++)
                if (is_subset(oldfinals, stnt[i])) *newfinals++ = i+'A';
        *newfinals++ = '\0';
}
int main()
{
        load_DFA_table();
        print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);

        N_optDFA_states = optimize_DFA(DFAtab, N_DFA_states,
                                N_symbols, DFA_finals, StateName, OptDFA);
        get_NEW_finals(NEW_finals, DFA_finals, StateName, N_optDFA_states);

        print_dfa_table(OptDFA, N_optDFA_states, N_symbols, NEW_finals);

        return 0;
}
```

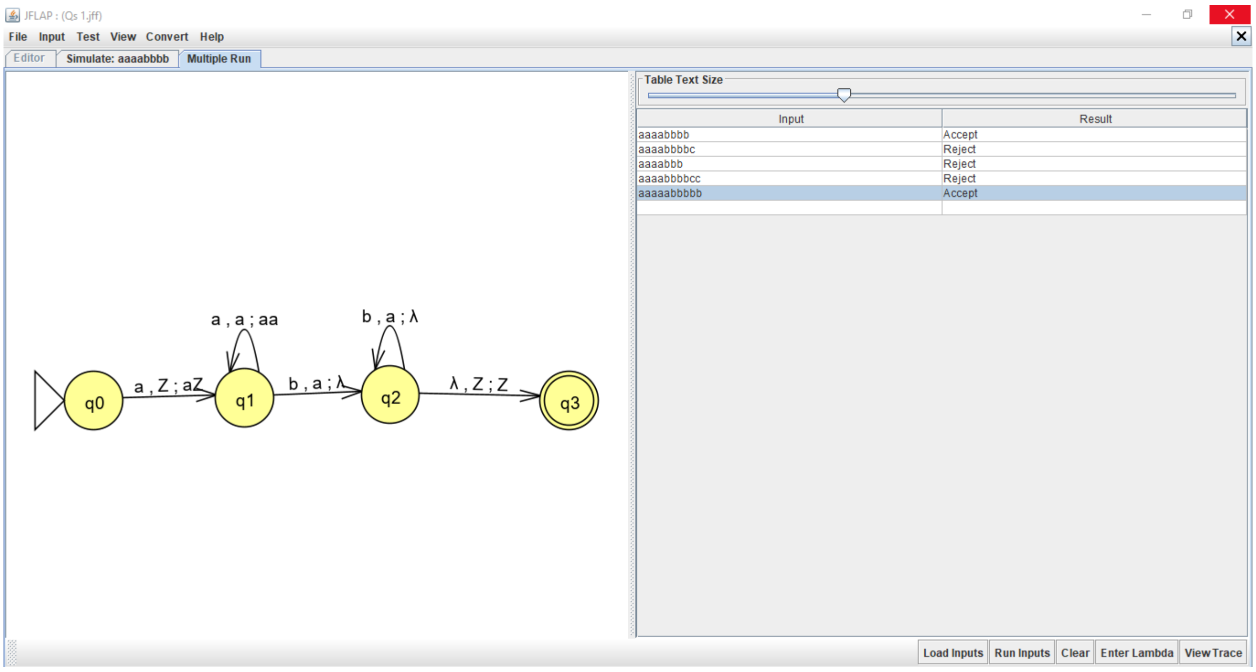
Output:



```
DFA: STATE TRANSITION TABLE
     |   0    1
-----+-----------
  A  |   B    C
  B  |   E    F
  C  |   A    A
  D  |   F    E
  E  |   D    F
  F  |   D    E
Final states = EF

EQUIV. CLASS CANDIDATE ==> 0:[ABCD] 1:[EF]
   0:[ABCD]     --> [BEAF] (0101)
   0:[ABCD]     --> [CFAE] (0101)
   1:[EF]       --> [DD] (00)
   1:[EF]       --> [FE] (11)

EQUIV. CLASS CANDIDATE ==> 0:[AC] 1:[BD] 2:[EF]
   0:[AC]       --> [BA] (10)
   0:[AC]       --> [CA] (00)
   1:[BD]       --> [EF] (22)
   1:[BD]       --> [FE] (22)
   2:[EF]       --> [DD] (11)
   2:[EF]       --> [FE] (22)

EQUIV. CLASS CANDIDATE ==> 0:[A] 1:[BD] 2:[C] 3:[EF]
   0:[A]        --> [B] (1)
   0:[A]        --> [C] (2)
   1:[BD]       --> [EF] (33)
   1:[BD]       --> [FE] (33)
   2:[C]        --> [A] (0)
   2:[C]        --> [A] (0)
   3:[EF]       --> [DD] (11)
   3:[EF]       --> [FE] (33)

DFA: STATE TRANSITION TABLE
     |   0    1
-----+-----------
  A  |   B    C
  B  |   D    D
  C  |   A    A
  D  |   B    D
Final states = D

Process returned 0 (0x0)   execution time : 0.200 s
Press any key to continue.
```

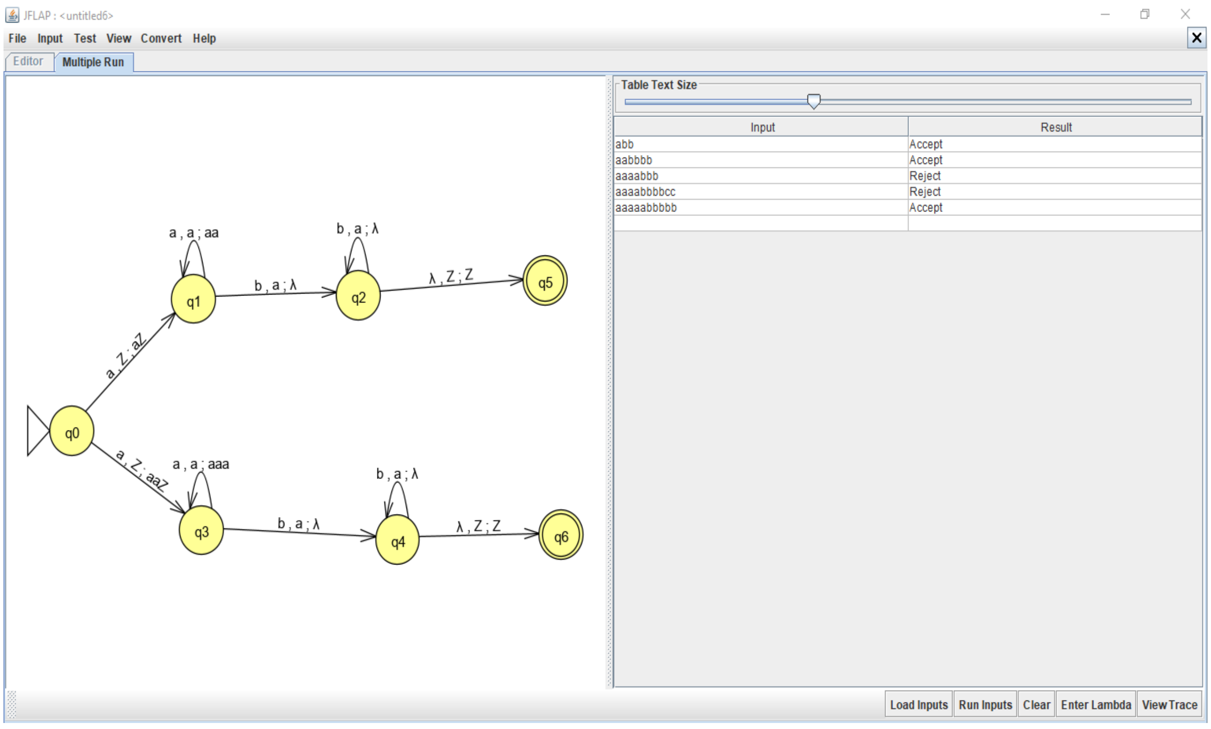
Q2. Questions given using JFLAP.

a).



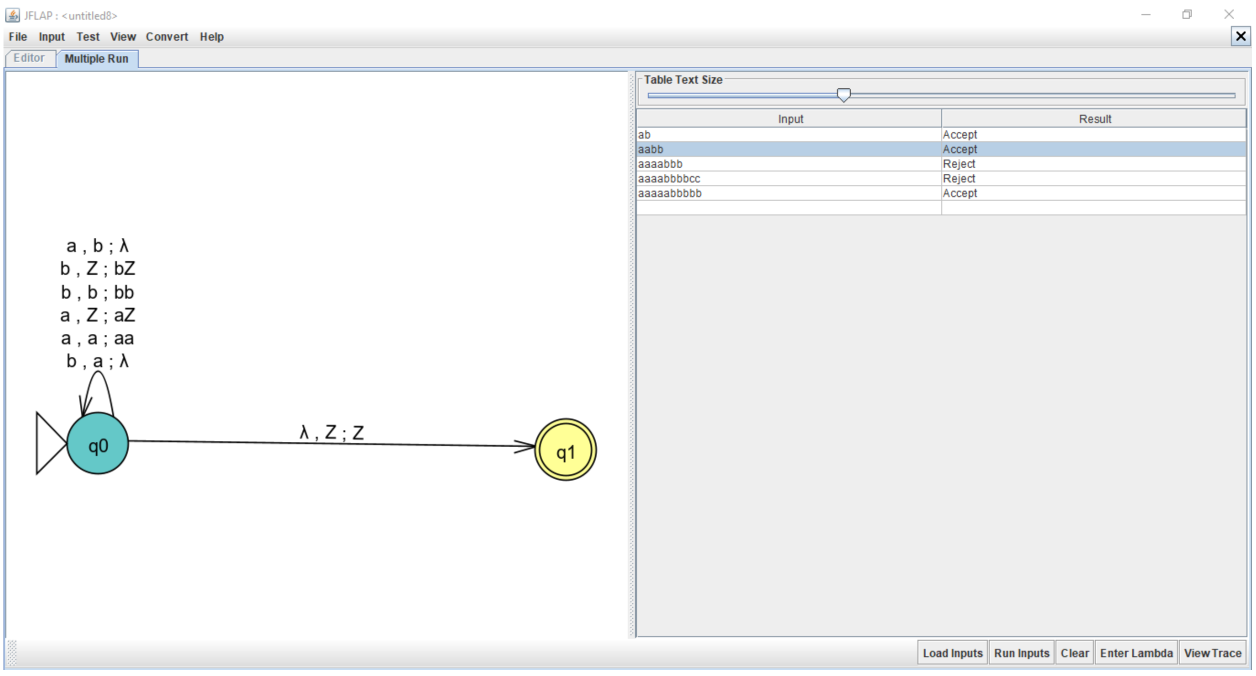
Input	Result
aaaabbbb	Accept
aaaabbbbc	Reject
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Accept

b).



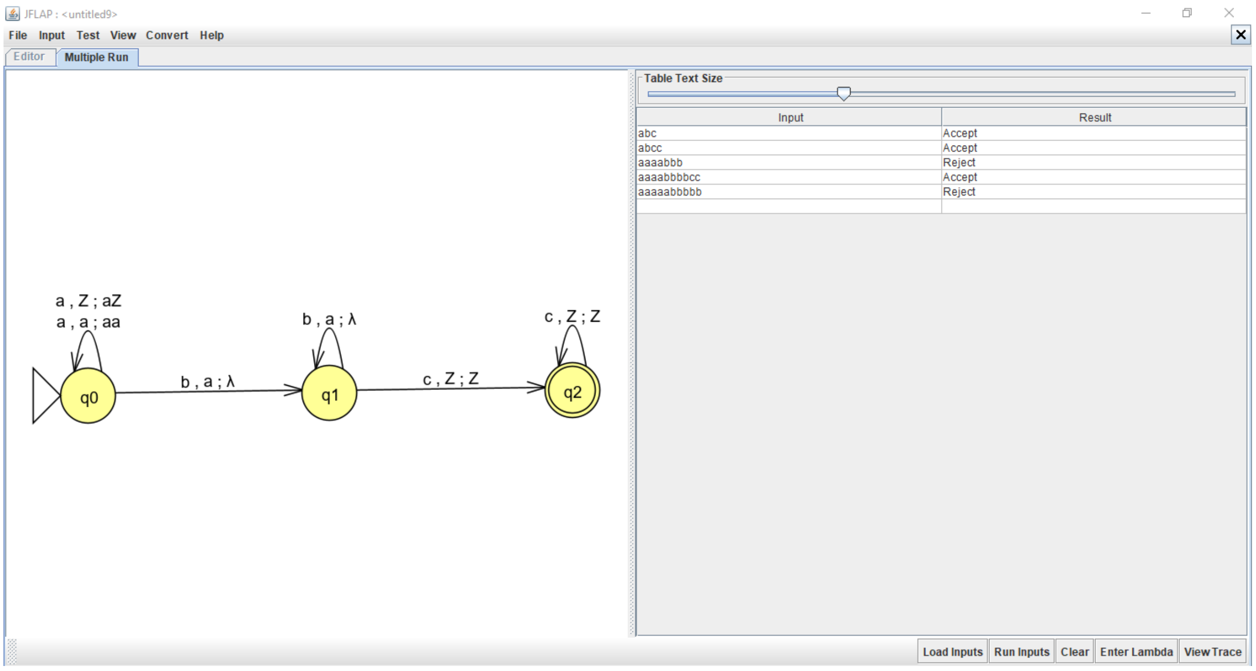
Input	Result
abb	Accept
aabbbb	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbbb	Accept

c).



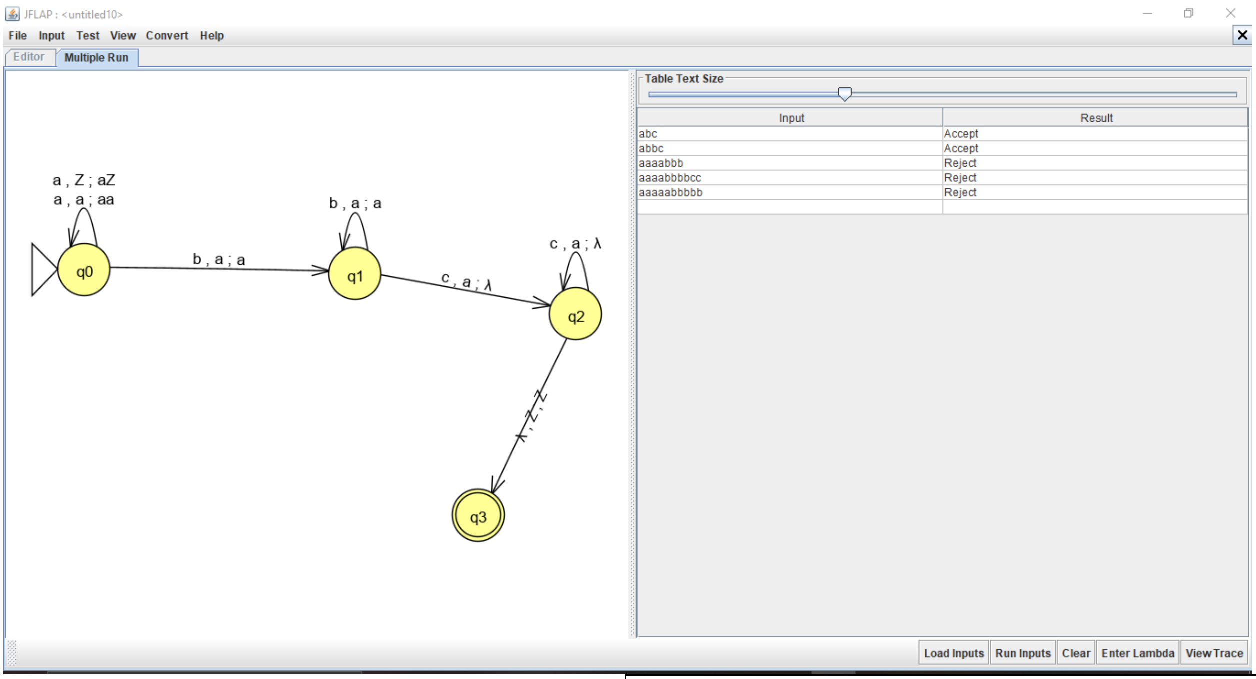
Input	Result
ab	Accept
aabb	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Accept

d).



Input	Result
abc	Accept
abcc	Accept
aaaabbb	Reject
aaaabbbbcc	Accept
aaaaabbbbb	Reject

e).



Input	Result
abc	Accept
abbc	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Reject

Name:Akash Kumar

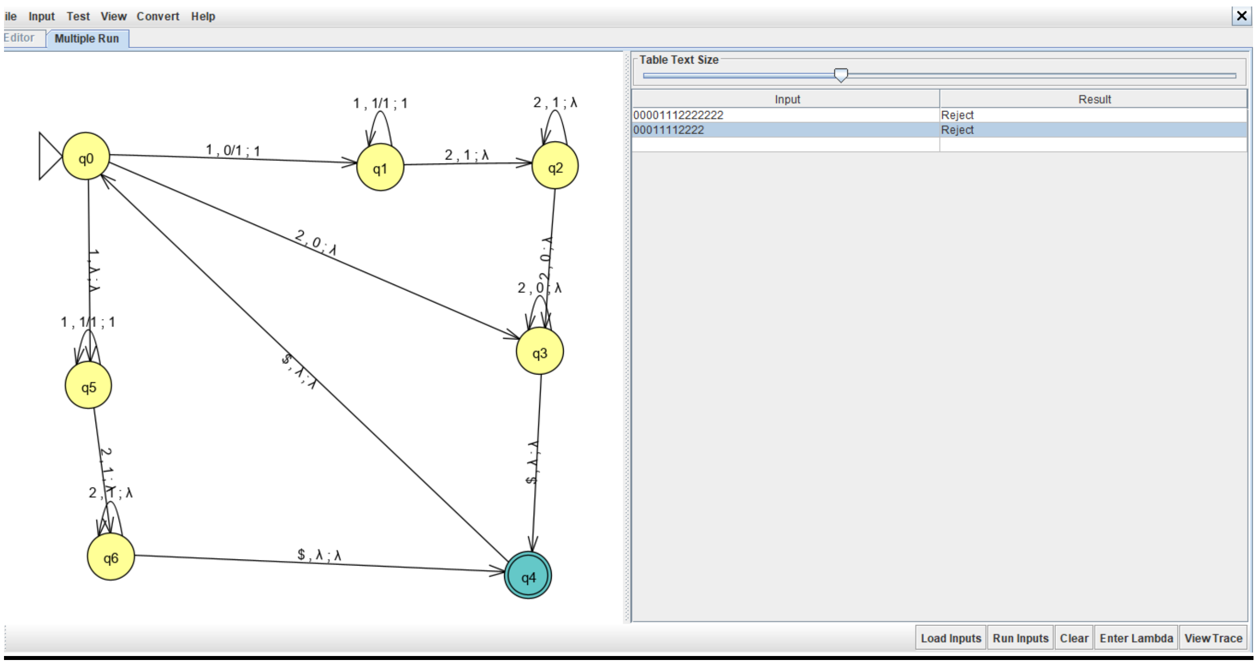
RegNo -1801289021

RollNo-06

Branch-CSE-A

ASSIGNMENT-7

Q1.WAP for design and implementation of PDA that accepts the language {(0^n) (1^n) | n>=0}.

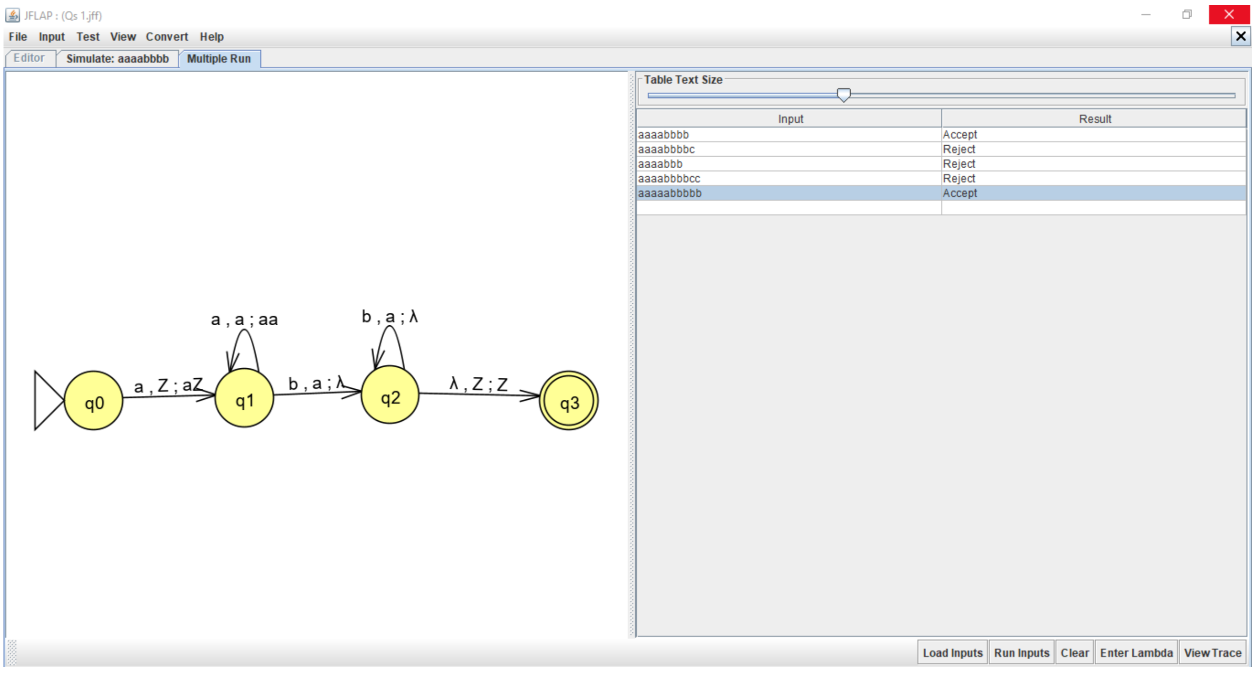


Input	Result
00001112222222	Reject
00011112222	Reject

Q2. Questions given using JFLAP

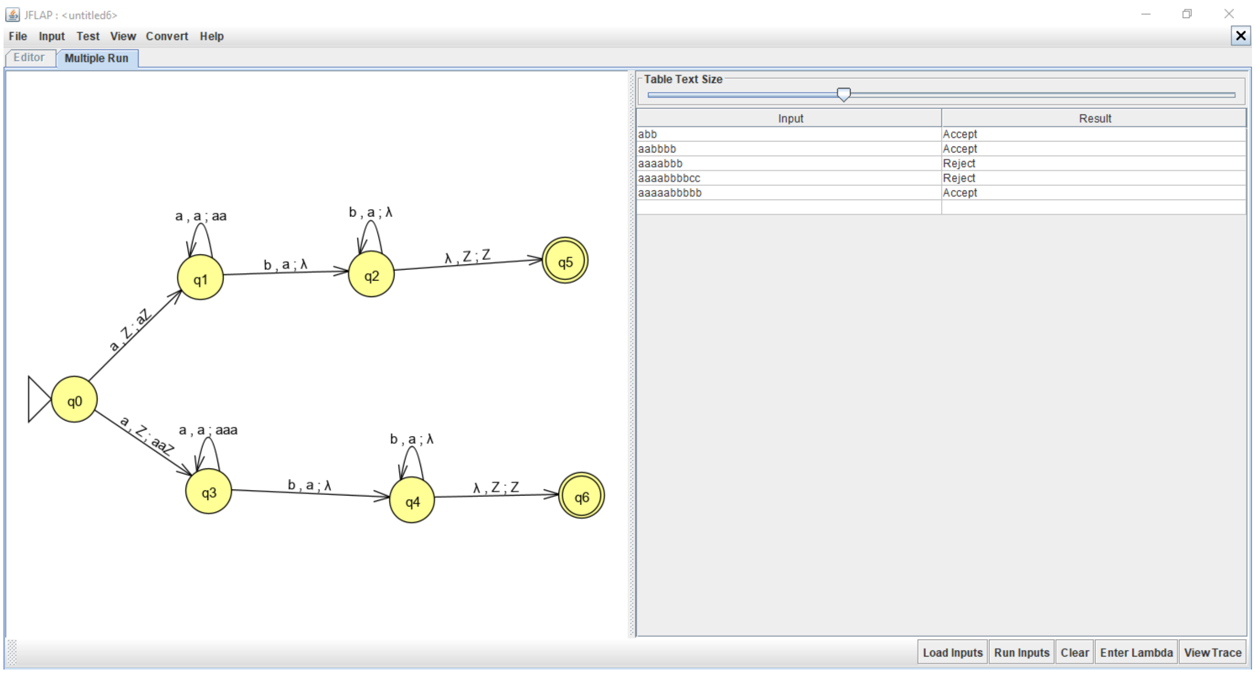
Construct a PDA over {a,b} accepting a language

a) L = {anbn | n>=1 }



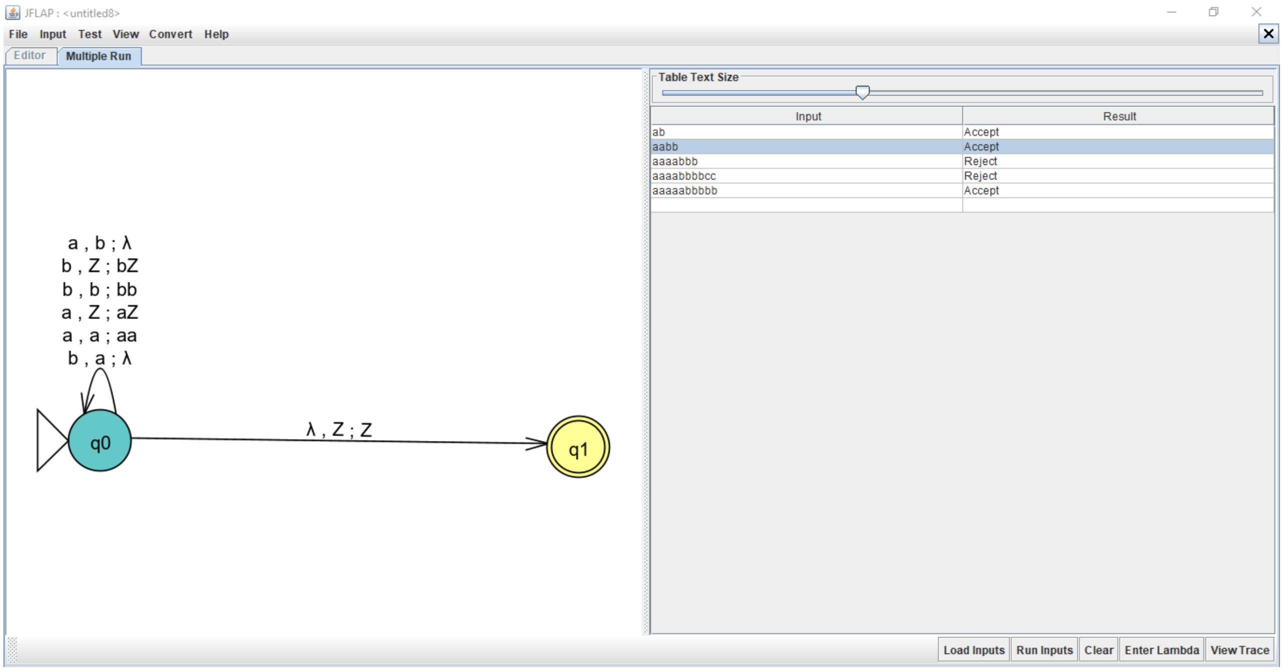
Input	Result
aaaabbbb	Accept
aaaabbbbc	Reject
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Accept

b) L = {w |n a(w) = n b(w).



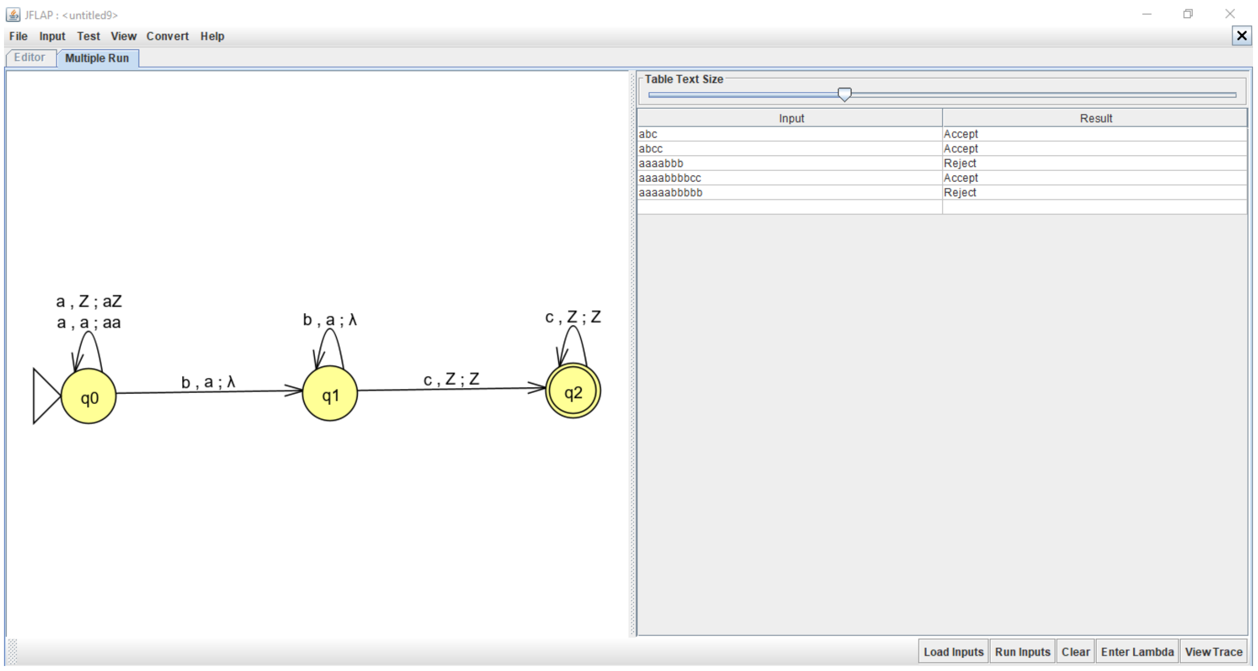
Input	Result
abb	Accept
aabbbb	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Accept

c) L = {anb2n | n>=1}



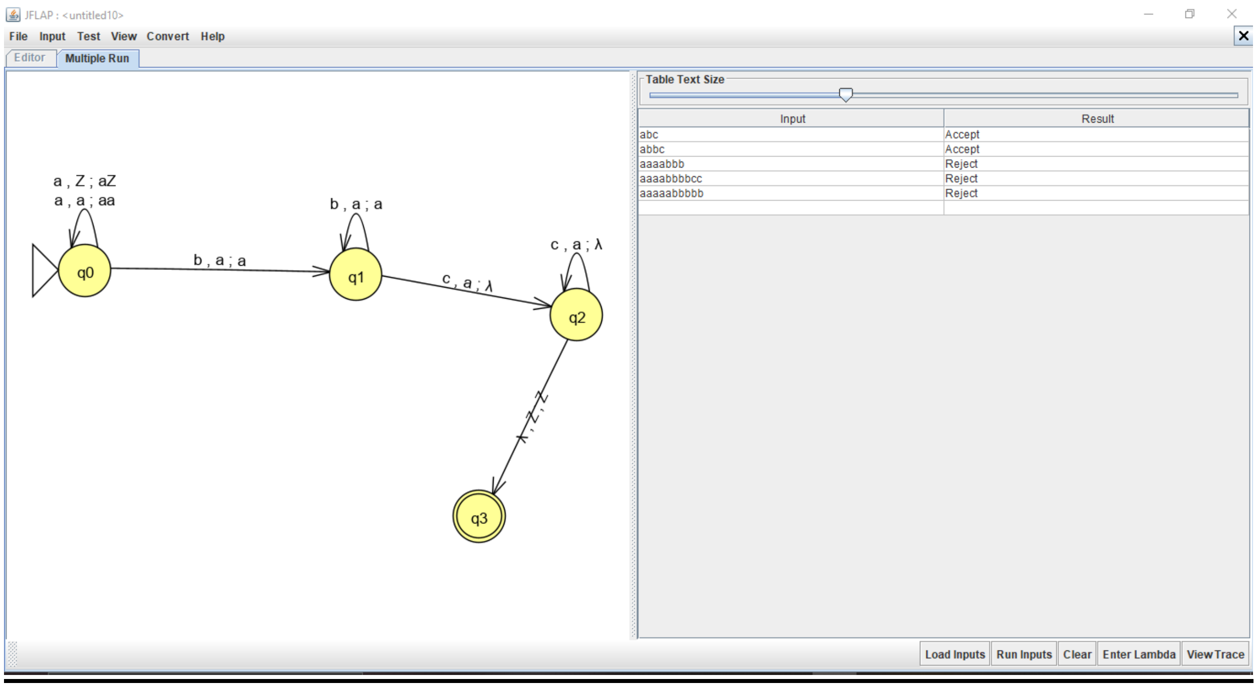
Input	Result
ab	Accept
aabb	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Accept

d). L= {anbn cm | n,m>=1 }



Input	Result
abc	Accept
abcc	Accept
aaaabbb	Reject
aaaabbbbcc	Accept
aaaaabbbbb	Reject

e). L = {anbm cn | n,m>=1 }



Input	Result
abc	Accept
abbc	Accept
aaaabbb	Reject
aaaabbbbcc	Reject
aaaaabbbbb	Reject

Name:Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A

ASSIGNMENT-8

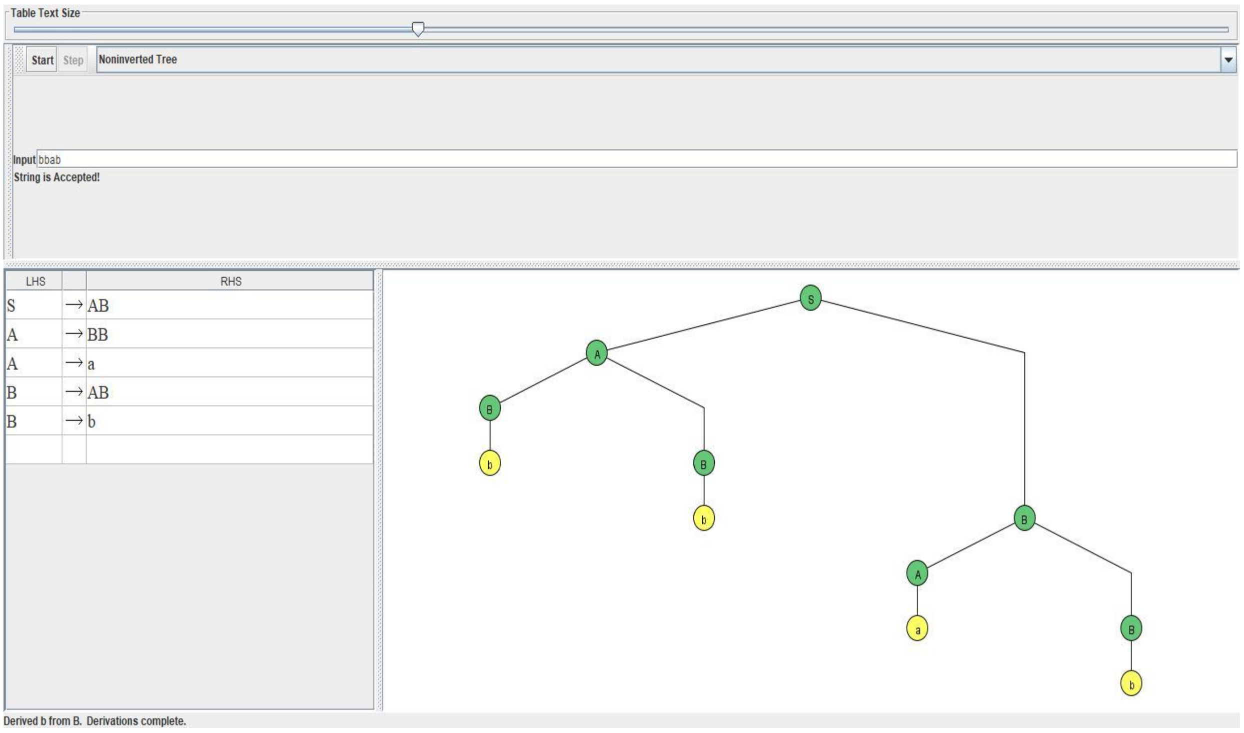
Q1.CYK parsing algorithm for some specific Context free grammars

using JFLAP.

A). S->AB/A

A->BB/a

B->AB/b



Input bbab

String is Accepted!

LHS		RHS
S	→	AB
A	→	BB
A	→	a
B	→	AB
B	→	b

Derived b from B. Derivations complete.

Name: Akash Kumar

RegNo -1801289021

RollNo-06

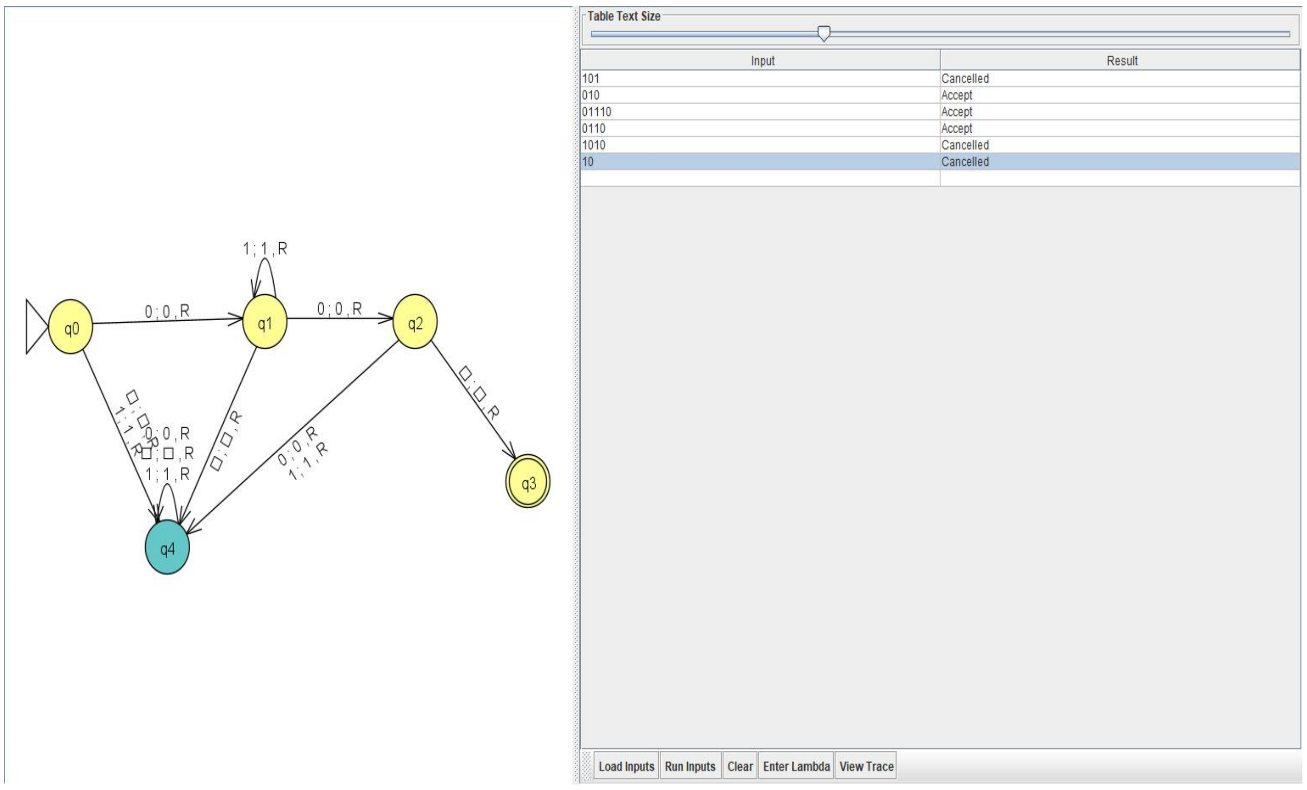
Branch-CSE-A

ASSIGNMENT-9

Q1. Turing machine for some Recursively Languages using JFLAP.

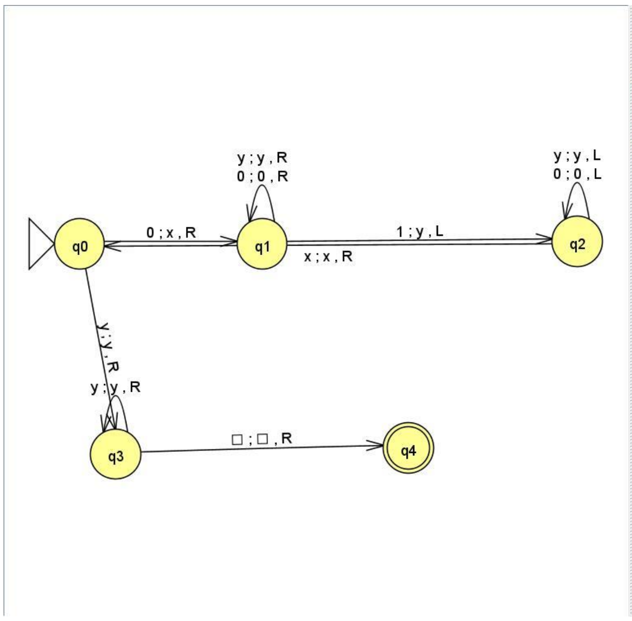
Design the TM that recognizes the language

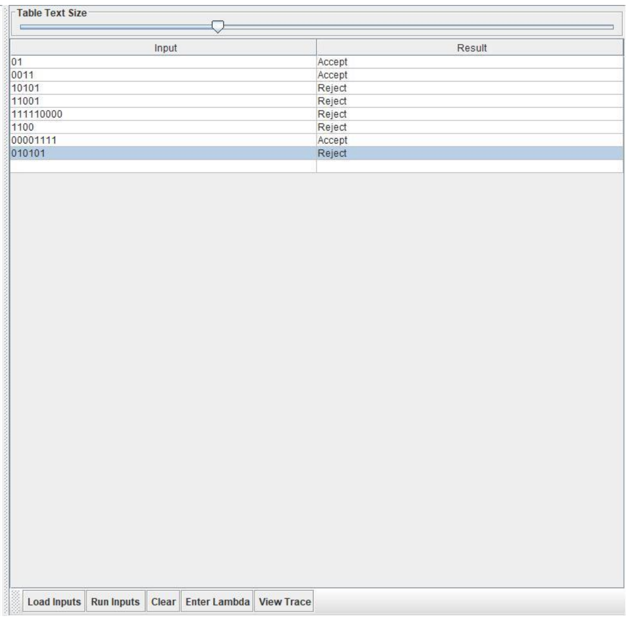
a) L = 01*0



Input	Result
101	Cancelled
010	Accept
01110	Accept
0110	Accept
1010	Cancelled
10	Cancelled

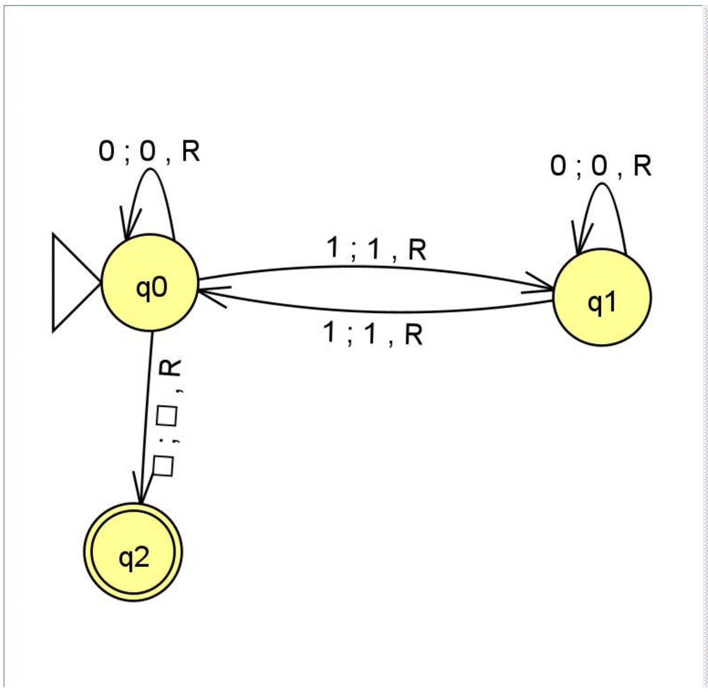
b).L = {0n1n | n>=0}

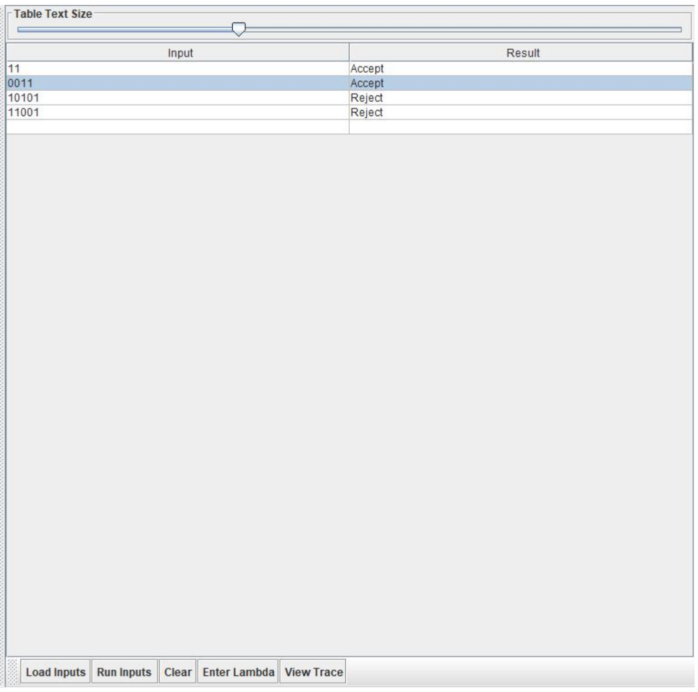




Input	Result
01	Accept
0011	Accept
10101	Reject
11001	Reject
111110000	Reject
1100	Reject
00001111	Accept
010101	Reject

C). L = {w| w ε {0,1}* and have even number of 1' s}





Input	Result
11	Accept
0011	Accept
10101	Reject
11001	Reject

d). Construct a TM over {a,b} accepting strings as even palindrome .

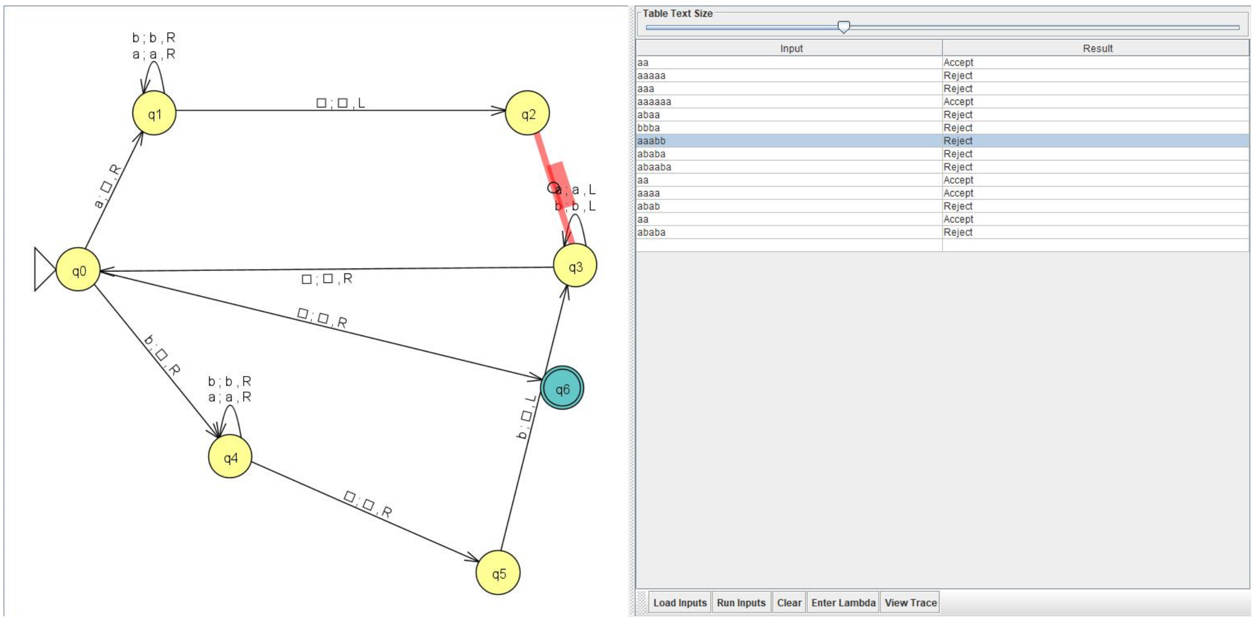


Table Text Size

Input	Result
aa	Accept
aaaaa	Reject
aaa	Reject
aaaaaa	Accept
abaa	Reject
bbba	Reject
aaabb	Reject
ababa	Reject
abaaba	Reject
aa	Accept
aaaa	Accept
abab	Reject
aa	Accept
ababa	Reject

e). L={anbncn | n>=0 }

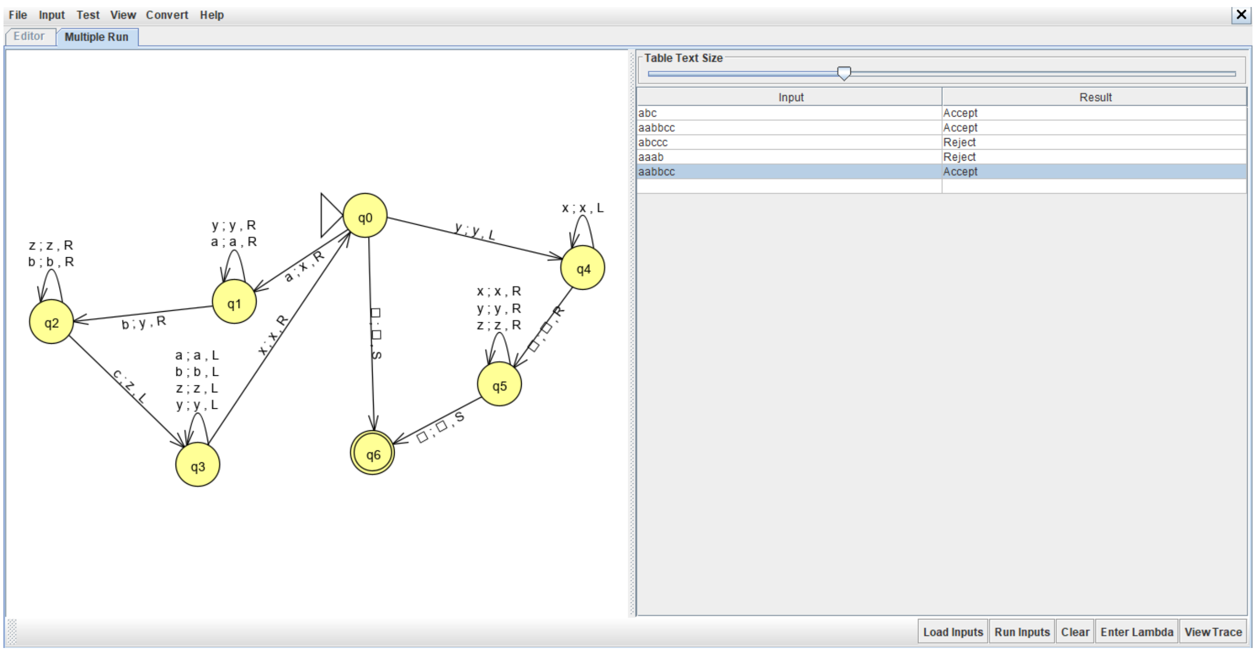


Table Text Size

Input	Result
abc	Accept
aabbcc	Accept
abccc	Reject
aaab	Reject
aabbcc	Accept

Name: Akash Kumar

RegNo -1801289021

RollNo-06

Branch-CSE-A