Compiler Design  
Lab Exercise 2

short line

**Aim: Conversion of a regular expression to NFA.**

**Algorithm:**

1. Start
2. Get the input from the user.
3. Initialize separate 2D array which tracks and modifies its contents according to the given regular expression.
4. Create separate parts for different operators
5. A regular expression is in the form of a.b (or) a+b.
6. Display the output
7. Compare with your table

**Code:(Language C++)**

#include<iostream>

#include<string.h>

int main()

{

    printf("Enter the regular expression: ");

    char reg[20];

    int q[20][3],i,j,len,a,b;

    for(a=0;a<20;a++)

    {

        for(b=0;b<3;b++)

        {

            q[a][b]=0;

        }

    }

    scanf("%s",reg);

    len=strlen(reg);

    i=0;

    j=1;

    while(i<len)

    {

        if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='\*')

        {

            q[j][0]=j+1;

            j++;

        }

        if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='\*')

        {

            q[j][1]=j+1;

            j++;

        }

        if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='\*')

        {

            q[j][2]=j+1;

            j++;

        }

        if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')

        {

            q[j][2]=((j+1)\*10)+(j+3);

            j++;

            q[j][0]=j+1;

            j++;

            q[j][2]=j+3;

            j++;

            q[j][1]=j+1;

            j++;

            q[j][2]=j+1;

            j++;

            i=i+2;

        }

        if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')

        {

            q[j][2]=((j+1)\*10)+(j+3);

            j++;

            q[j][1]=j+1;

            j++;

            q[j][2]=j+3;

            j++;

            q[j][0]=j+1;

            j++;

            q[j][2]=j+1;

            j++;

            i=i+2;

        }

        if(reg[i]=='a'&&reg[i+1]=='\*')

        {

            q[j][2]=((j+1)\*10)+(j+3);

            j++;

            q[j][0]=j+1;

            j++;

            q[j][2]=((j+1)\*10)+(j-1);

            j++;

        }

        if(reg[i]=='b'&&reg[i+1]=='\*')

        {

            q[j][2]=((j+1)\*10)+(j+3);

            j++;

            q[j][1]=j+1;

            j++;

            q[j][2]=((j+1)\*10)+(j-1);

            j++;

        }

        if(reg[i]==')'&&reg[i+1]=='\*')

        {

            q[0][2]=((j+1)\*10)+1;

            q[j][2]=((j+1)\*10)+1;

            j++;

        }

        i++;

    }

    printf("Transition function \n");

    for(i=0;i<=j;i++)

    {

        if(q[i][0]!=0)

            printf("\n q[%d,a]-->%d",i,q[i][0]);

        if(q[i][1]!=0)

            printf("\n q[%d,b]-->%d",i,q[i][1]);

        if(q[i][2]!=0)

        {

            if(q[i][2]<10)

                printf("\n q[%d,e]-->%d",i,q[i][2]);

            else

                printf("\n q[%d,e]-->%d & %d",i,q[i][2]/10,q[i][2]%10);

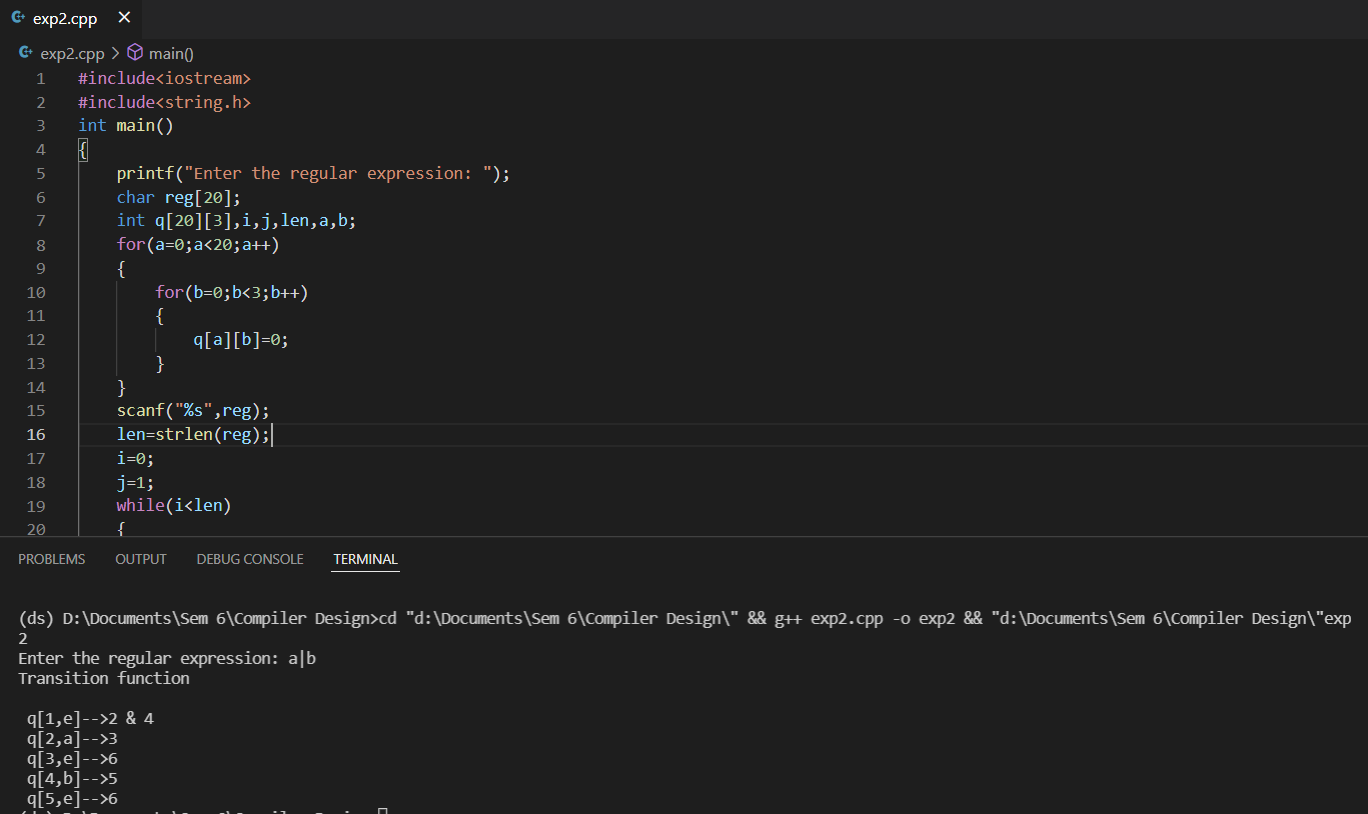
        }

    }

    return 0;

}

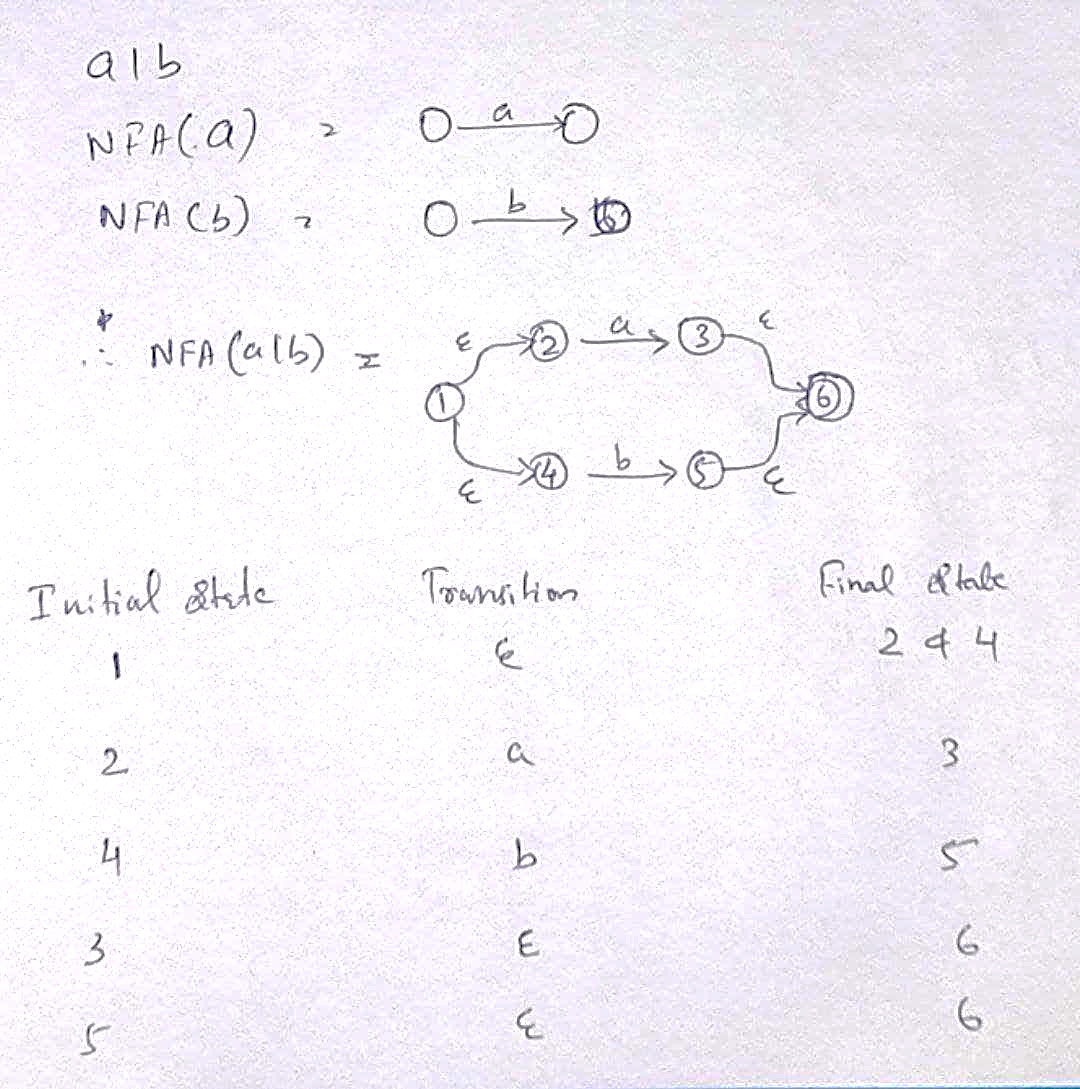
**Output:**



**Transition table:**

* First, we construct NFA for *a*
* Then we construct NFA for *b*
* After the above steps, we construct the NFA for *a/b* according to ***Thomson’s Construction*** method.

We compare the transition tables from code as well as handwritten and deduce that they are the same for the given expression.



**Conclusion:** Both the manual and output transition tables are the same hence the conversion of a regular expression to NFA is verified.