RA2011031010046

EXPERIMENT : 2

CONSTRUCTION OF NFA FROM REGULAR EXPRESSION

DATE : 01/02/23

AIM : Write a program in C to conduct a non-deterministic finite automata (NFA) from regular expression.

PROCEDURE:

The method followed is that we decompose the RE into the primitive components. For each component, we construct a finite automaton as follows:

1. For epsilon, we construct the NFA.
2. For a in language, we construct the NFA.
3. Having constructed the components for the basic regular expressions, we proceed to combine them in ways that correspond to the way compounded regular expressions are formed from small regular expressions.
4. For regular expressions R1/R2 we construct the composite NFA.
5. For R1,R2 we construct the composite NFA.
6. Create a single start state for the automation and mark it as the initial state.
7. For each character in the regular expression, create a new state and add an edge between the previous state and the new state, with the character as the label.
8. For each operator in the regular expression create a new state and add the appropriate edges to represent the operator.
9. Mark the final state the accepting state, which in the state that is reached when the regular expression is fully matched.
10. Stop the program.

PROGRAM:

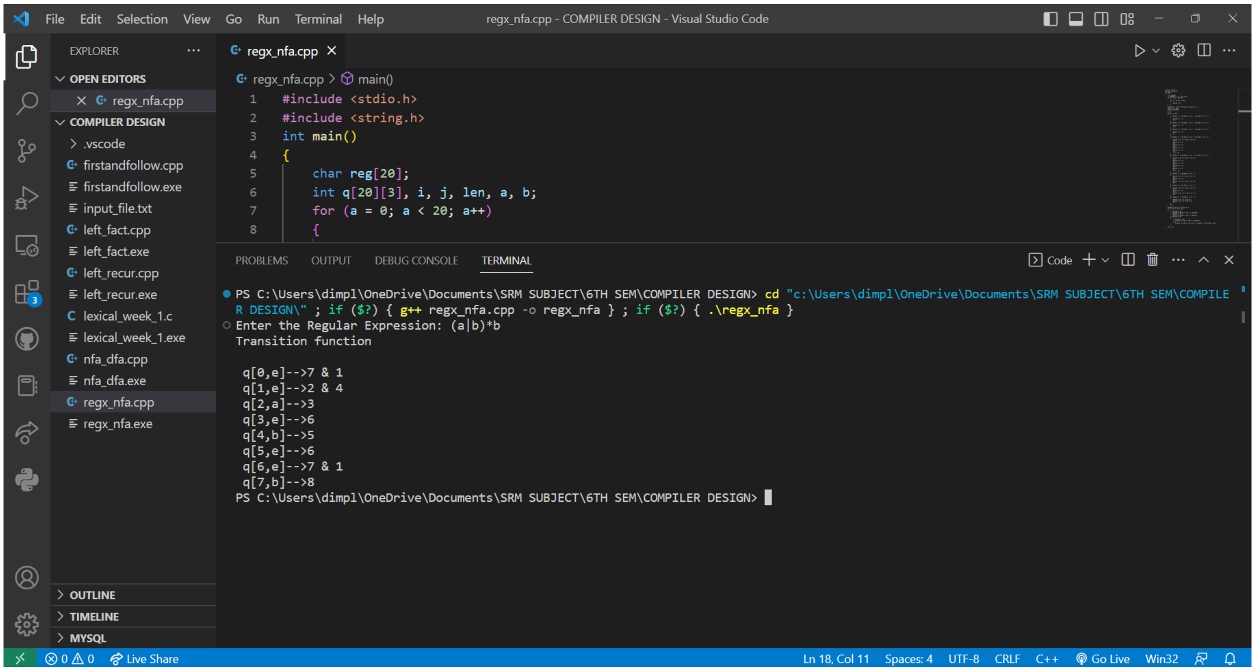
```
#include <stdio.h>
#include <string.h>
int main()
{
   char reg[20];
   int q[20][3], i, j, len, a, b;
   for (a = 0; a < 20; a++)
   {
      for (b = 0; b < 3; b++)
      {
         q[a][b] = 0;
      }
   }
   printf("%s", "Enter the Regular Expression: ");
   scanf("%s", reg);
   len = strlen(reg);
   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++;
    }
    // 1
    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;
}
```

INPUT:

(a|b)*b

OUTPUT:



```
#include <stdio.h>
#include <string.h>
int main()
{
    char reg[20];
    int q[20][3], i, j, len, a, b;
    for (a = 0; a < 20; a++)
    {
```

```
PS C:\Users\dimpl\OneDrive\Documents\SRM SUBJECT\6TH SEM\COMPILER DESIGN> cd "c:\Users\dimpl\OneDrive\Documents\SRM SUBJECT\6TH SEM\COMPILE
R DESIGN\" ; if ($?) { g++ regx_nfa.cpp -o regx_nfa } ; if ($?) { .\regx_nfa }
Enter the Regular Expression: (a|b)*b
Transition function

 q[0,e]-->7 & 1
 q[1,e]-->2 & 4
 q[2,a]-->3
 q[3,e]-->6
 q[4,b]-->5
 q[5,e]-->6
 q[6,e]-->7 & 1
 q[7,b]-->8
PS C:\Users\dimpl\OneDrive\Documents\SRM SUBJECT\6TH SEM\COMPILER DESIGN>
```

RESULT:

Hence, the conversion of Regular Expression to NFA was successfully implemented.