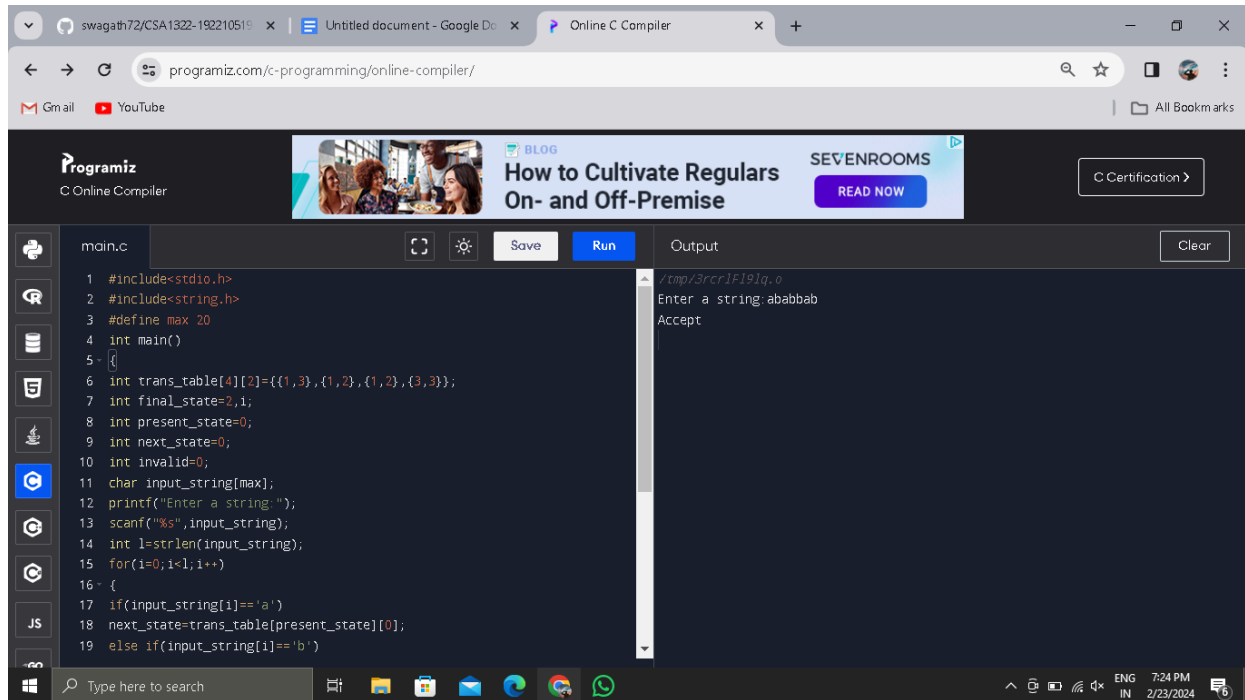


1.DETERMINISTIC FINITE AUTOMATA (DFA)

Code:

```
#include<stdio.h>
#include<string.h>
#define max 20
int main()
{
int trans_table[4][2]={{1,3},{1,2},{1,2},{3,3}};
int final_state=2,i;
int present_state=0;
int next_state=0;
int invalid=0;
char input_string[max];
printf("Enter a string:");
scanf("%s",input_string);
int l=strlen(input_string);
for(i=0;i<l;i++)
{
if(input_string[i]=='a')
next_state=trans_table[present_state][0];
else if(input_string[i]=='b')
next_state=trans_table[present_state][1];
else
invalid=1;
present_state=next_state;
}
if(invalid==1)
{
printf("Invalid input");
}
else if(present_state==final_state)
printf("Accept\n");
else
printf("Don't Accept\n");
}
```

Execution:



2. NON-DETERMINISTIC FINITE AUTOMATA (NFA)

Code:

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

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

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

```
int main() {
    int i, j, k, l, m, next_state[20], n, mat[10][10][10], flag, p;
    int num_states, final_state[5], num_symbols, num_final;
    int present_state[20], prev_trans, new_trans;
    char ch, input[20];
    int symbol[5], inp, inp1;
```

```
    printf("How many states in the NFA : ");
    scanf("%d", &num_states);
```

```
    printf("How many symbols in the input alphabet : ");
    scanf("%d", &num_symbols);
```

```
    for (i = 0; i < num_symbols; i++) {
        printf("Enter the input symbol %d : ", i + 1);
        scanf("%d", &symbol[i]);
```

```

}

printf("How many final states : ");
scanf("%d", &num_final);

for (i = 0; i < num_final; i++) {
    printf("Enter the final state %d : ", i + 1);
    scanf("%d", &final_state[i]);
}

// Initialize all entries with -1 in Transition table
for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        for (k = 0; k < 10; k++) {
            mat[i][j][k] = -1;
        }
    }
}

// Get input from the user and fill the 3D transition table
for (i = 0; i < num_states; i++) {
    for (j = 0; j < num_symbols; j++) {
        printf("How many transitions from state %d for the input %d: ", i, symbol[j]);
        scanf("%d", &n);
        for (k = 0; k < n; k++) {
            printf("Enter the transition %d from state %d for the input %d: ", k + 1, i, symbol[j]);
            scanf("%d", &mat[i][j][k]);
        }
    }
}

printf("The transitions are stored as shown below\n");

for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        for (k = 0; k < 10; k++) {
            if (mat[i][j][k] != -1)
                printf("mat[%d][%d][%d] = %d\n", i, j, k, mat[i][j][k]);
        }
    }
}

while (1) {
    printf("Enter the input string : ");

```

```

scanf("%s", input);

present_state[0] = 0;
prev_trans = 1;
l = strlen(input);

for (i = 0; i < l; i++) {
    if (input[i] == '0')
        inp1 = 0;
    else if (input[i] == '1')
        inp1 = 1;
    else {
        printf("Invalid input\n");
        exit(0);
    }

    for (m = 0; m < num_symbols; m++) {
        if (inp1 == symbol[m]) {
            inp = m;
            break;
        }
    }

    new_trans = 0;

    for (j = 0; j < prev_trans; j++) {
        k = 0;
        p = present_state[j];
        while (mat[p][inp][k] != -1) {
            next_state[new_trans++] = mat[p][inp][k];
            k++;
        }
    }

    for (j = 0; j < new_trans; j++) {
        present_state[j] = next_state[j];
    }

    prev_trans = new_trans;
}

flag = 0;

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

```

```

        for (j = 0; j < num_final; j++) {
            if (present_state[i] == final_state[j]) {
                flag = 1;
                break;
            }
        }
    }

    if (flag == 1)
        printf("Accepted\n");
    else
        printf("Not accepted\n");

    printf("Try with another input\n");
}

return 0;
}

```

Execution:

The screenshot displays the Programiz online C compiler. The code in the editor is as follows:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main() {
6     int i, j, k, l, m, next_state[20], n, mat[10][10][10], flag, p;
7     int num_states, final_state[5], num_symbols, num_final;
8     int present_state[20], prev_trans, new_trans;
9     char ch, input[20];
10    int symbol[5], inp, inpl;
11
12    printf("How many states in the NFA : ");
13    scanf("%d", &num_states);
14
15    printf("How many symbols in the input alphabet : ");
16    scanf("%d", &num_symbols);
17
18    for (i = 0; i < num_symbols; i++) {
19        printf("Enter the input symbol %d : ", i + 1);
20        scanf("%d", &symbol[i]);
21    }
22
23    printf("How many final states : ");
24    scanf("%d", &num_final);

```

The output window shows the following execution steps:

```

How many transitions from state 1 for the input 1: 1
Enter the transition 1 from state 1 for the input 1: 1
How many transitions from state 2 for the input 0: 0
How many transitions from state 2 for the input 1: 0
How many transitions from state 3 for the input 0: 1
Enter the transition 1 from state 3 for the input 0: 3
How many transitions from state 3 for the input 1: 2
Enter the transition 1 from state 3 for the input 1: 2
Enter the transition 2 from state 3 for the input 1: 3
The transitions are stored as shown below
mat[0][0][0] = 1
mat[0][1][0] = 3
mat[1][0][0] = 1
mat[1][0][1] = 2
mat[1][1][0] = 1
mat[3][0][0] = 3
mat[3][1][0] = 2
mat[3][1][1] = 3
Enter the input string : 1010000101
Accepted
Try with another input
Enter the input string : 10101000110
Not accepted
Try with another input
Enter the input string :

```

3.FINDING ϵ -CLOSURE FOR NFA WITH ϵ -MOVES

Code:

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

```

#include <string.h>

int trans_table[10][5][3];
char symbol[5];
int e_closure[10][10], ptr, state;

void find_e_closure(int x);

int main() {
    int i, j, k, n, num_states, num_symbols;

    for (i = 0; i < 10; i++) {
        for (j = 0; j < 5; j++) {
            for (k = 0; k < 3; k++) {
                trans_table[i][j][k] = -1;
            }
        }
    }

    printf("How many states in the NFA with e-moves: ");
    scanf("%d", &num_states);

    printf("How many symbols in the input alphabet including e: ");
    scanf("%d", &num_symbols);

    printf("Enter the symbols without space. Give 'e' first: ");
    scanf("%s", symbol);

    for (i = 0; i < num_states; i++) {
        for (j = 0; j < num_symbols; j++) {
            printf("How many transitions from state %d for the input %c: ", i, symbol[j]);
            scanf("%d", &n);
            for (k = 0; k < n; k++) {
                printf("Enter the transition %d from state %d for the input %c: ", k + 1, i, symbol[j]);
                scanf("%d", &trans_table[i][j][k]);
            }
        }
    }

    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
            e_closure[i][j] = -1;
        }
    }
}

```

```

for (i = 0; i < num_states; i++)
    e_closure[i][0] = i;

for (i = 0; i < num_states; i++) {
    if (trans_table[i][0][0] == -1)
        continue;
    else {
        state = i;
        ptr = 1;
        find_e_closure(i);
    }
}

for (i = 0; i < num_states; i++) {
    printf("e-closure(%d) = {", i);
    for (j = 0; j < num_states; j++) {
        if (e_closure[i][j] != -1) {
            printf("%d, ", e_closure[i][j]);
        }
    }
    printf("}\n");
}

return 0;
}

void find_e_closure(int x) {
    int i, j, y[10], num_trans;
    i = 0;

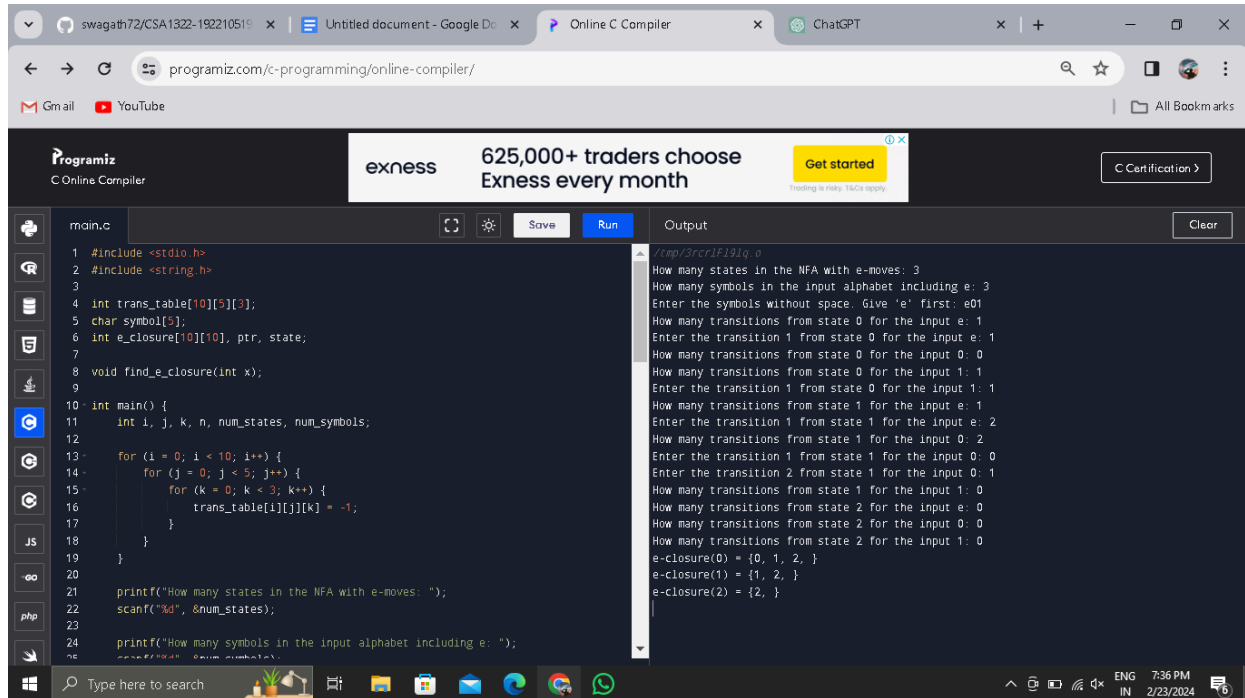
    while (trans_table[x][0][i] != -1) {
        y[i] = trans_table[x][0][i];
        i = i + 1;
    }

    num_trans = i;

    for (j = 0; j < num_trans; j++) {
        e_closure[state][ptr] = y[j];
        ptr++;
        find_e_closure(y[j]);
    }
}

```

Execution:



The screenshot shows a web browser with multiple tabs, including 'Online C Compiler'. The compiler interface has a dark theme. On the left, a file explorer shows 'main.c'. The main editor contains C code for finding the e-closure of states in an NFA. The 'Run' button is highlighted. On the right, the 'Output' pane shows the program's execution results, including prompts for the number of states, symbols, and transitions, followed by the calculated e-closures for states 0, 1, and 2.

```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int trans_table[10][5][3];
5 char symbol[5];
6 int e_closure[10][10], ptr, state;
7
8 void find_e_closure(int x);
9
10 int main() {
11     int i, j, k, n, num_states, num_symbols;
12
13     for (i = 0; i < 10; i++) {
14         for (j = 0; j < 5; j++) {
15             for (k = 0; k < 3; k++) {
16                 trans_table[i][j][k] = -1;
17             }
18         }
19     }
20
21     printf("How many states in the NFA with e-moves: ");
22     scanf("%d", &num_states);
23
24     printf("How many symbols in the input alphabet including e: ");
25     scanf("%d", &num_symbols);
26 }
```

Output

```
//tmp/3rcr1F191q.o
How many states in the NFA with e-moves: 3
How many symbols in the input alphabet including e: 3
Enter the symbols without space. Give 'e' first: e01
How many transitions from state 0 for the input e: 1
Enter the transition 1 from state 0 for the input e: 1
How many transitions from state 0 for the input 0: 0
How many transitions from state 0 for the input 1: 1
Enter the transition 1 from state 0 for the input 1: 1
How many transitions from state 1 for the input e: 1
Enter the transition 1 from state 1 for the input e: 2
How many transitions from state 1 for the input 0: 2
Enter the transition 1 from state 1 for the input 0: 0
Enter the transition 2 from state 1 for the input 0: 1
How many transitions from state 1 for the input 1: 0
How many transitions from state 2 for the input e: 0
How many transitions from state 2 for the input 0: 0
How many transitions from state 2 for the input 1: 0
e-closure(0) = {0, 1, 2, }
e-closure(1) = {1, 2, }
e-closure(2) = {2, }
```

4.CHECKING WHETHER A STRING BELONGS TO A GRAMMAR

Code:

```
#include<stdio.h>
#include<string.h>
int main(){
char s[100];
int i,flag;
int l;
printf("enter a string to check:");
scanf("%s",s);
l=strlen(s);
flag=1;
for(i=0;i<l;i++)
{
if(s[i]!='0' && s[i]!='1')
{
flag=0;
}
}
if(flag!=1)
printf("string is Not Valid\n");
```



```

if(flag==1)
{
if (s[0]=='0'&& s[l-1]=='1')
printf("string is accepted\n");
else
printf("string is Not accepted\n");
}
}

```

Execution:

The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The browser's address bar and tabs are visible at the top. The main content area is the Programiz C Online Compiler interface. On the left, there is a sidebar with icons for file management and a list of languages (C, JS, go, php) with 'C' selected. The central editor displays a C program in `main.c` with line numbers 4 through 27. The code implements a string validation logic: it checks if the first character is '0' and the last character is '1'. If both conditions are met, it prints "string is accepted"; otherwise, it prints "string is Not Valid". The code also includes a loop to calculate the string length and a flag to track validation. The 'Run' button is highlighted in blue. To the right of the editor is the 'Output' panel, which shows the execution results: "enter a string to check:01010111101" followed by "string is accepted". A 'Clear' button is located at the top right of the output panel. At the bottom of the browser window, the Windows taskbar is visible, showing the search bar, task icons, and system tray with the date and time (7:39 PM, 2/23/2024).

```

main.c
4  char s[100],
5  int i,flag;
6  int l;
7  printf("enter a string to check:");
8  scanf("%s",s);
9  l=strlen(s);
10 flag=1;
11 for(i=0;i<l;i++)
12 {
13 if(s[i]!='0' && s[i]!='1')
14 {
15 flag=0;
16 }
17 }
18 if(flag==1)
19 printf("string is Not Valid\n");
20 if(flag==1)
21 {
22 if (s[0]=='0' && s[l-1]=='1')
23 printf("string is accepted\n");
24 else
25 printf("string is Not accepted\n");
26 }
27 }

```

Output

```

/tmp/3rcrlF191q.o
enter a string to check:01010111101
string is accepted

```