**Домашнее задание №1.**

**Проектирование и реализация конечного распознавателя.**

*Вариант 1.22*.

Выполнил: Кравченко Александр Андреевич  
КТбо1-6

1. **Постановка задачи:**
   1. Построить детерминированный конечный автомат с входным алфавитом {a, b, c}, допускающий множество цепочек, начинающихся и заканчивающихся симметричной парой символов.
   2. Разработать программу, реализующую разработанный конечный распознаватель
2. **Словесное описание автомата:**
3. **Описание состояний автомата:**

q0 – начальное состояние

q1 – если первый символ пары – a(N);

q2 – если первый символ пары – b(N);

q3 – если первый символ пары – c(N);

q4 – если второй символ пары, начинающейся с ‘a’ – ‘a’(Y). Если последующий символ также будет ‘a’, автомат останется в этом состоянии;

q5 – если следующий символ в последовательности по ветке пары ‘aa’ – ‘b’(N);

q6 – если следующий символ в последовательности по ветке пары ‘aa’ – ‘c’(N);

q7 – если следующий символ в последовательности по ветке пары ‘aa’ – ‘a’(N);

q8 – если второй или следующий символ пары, начинающейся с ‘a’ – ‘b’(N);

q9 – если следующий символ в последовательности по ветке пары ‘ab’, после символа ‘b’ – ‘a’(Y);

q10 – если следующий символ в последовательности по ветке пары ‘ab’ – ‘a’(N);

q11 - если следующий символ в последовательности по ветке пары ‘ab’ – ‘c’(N);

q12 – если второй или следующий символ пары, начинающейся с ‘a’ – ‘c’(N);

q13 – если следующий символ в последовательности по ветке пары ‘ac’, после символа ‘c’ – ‘a’(Y);

q14 – если следующий символ в последовательности по ветке пары ‘ac’ – ‘a’(N);

q15 - если следующий символ в последовательности по ветке пары ‘ac’ – ‘b’(N);

q16 – если второй символ пары, начинающейся с ‘b’ – ‘b’(Y). Если последующий символ также будет ‘b’, автомат останется в этом состоянии;

q17 – если следующий символ в последовательности по ветке пары ‘bb’ – ‘a’(N);

q18 – если следующий символ в последовательности по ветке пары ‘bb’ – ‘c’(N);

q19 – если следующий символ в последовательности по ветке пары ‘bb’ – ‘b’(N);

q20 – если второй или следующий символ пары, начинающейся с ‘b’ – ‘a’(N);

q21 – если следующий символ в последовательности по ветке пары ‘ba’, после символа ‘a’ – ‘b’(Y);

q22 – если следующий символ в последовательности по ветке пары ‘ba’ – ‘b’(N);

q23 - если следующий символ в последовательности по ветке пары ‘ba’ – ‘c’(N);

q24 – если второй или следующий символ пары, начинающейся с ‘b’ – ‘c’(N);

q25 – если следующий символ в последовательности по ветке пары ‘bc’, после символа ‘c’ – ‘b’(Y);

q26 – если следующий символ в последовательности по ветке пары ‘bc’ – ‘b’(N);

q27 - если следующий символ в последовательности по ветке пары ‘bc’ – ‘a’(N);

q28 – если второй символ пары, начинающейся с ‘c’ – ‘c’(Y). Если последующий символ также будет ‘c’, автомат останется в этом состоянии;

q29 – если следующий символ в последовательности по ветке пары ‘cc’ – ‘b’(N);

q30 – если следующий символ в последовательности по ветке пары ‘cc’ – ‘a’(N);

q31 – если следующий символ в последовательности по ветке пары ‘cc’ – ‘c’(N);

q32– если второй или следующий символ пары, начинающейся с ‘c’ – ‘b’(N);

q33 – если следующий символ в последовательности по ветке пары ‘cb’, после символа ‘b’ – ‘c’(Y);

q34 – если следующий символ в последовательности по ветке пары ‘cb’ – ‘c’(N);

q35 - если следующий символ в последовательности по ветке пары ‘cb’ – ‘a’(N);

q36– если второй или следующий символ пары, начинающейся с ‘c’ – ‘a’(N);

q37 – если следующий символ в последовательности по ветке пары ‘ca’, после символа ‘a’ – c’(Y);

q38 – если следующий символ в последовательности по ветке пары ‘ca’ – ‘a’(N);

q39 - если следующий символ в последовательности по ветке пары ‘ca’ – ‘b’(N);

1. **Общая логика работы автомата:**

Общая логика работы автомата состоит в том, для каждой возможной пары элементов есть своя ветка(свой блок). Всего 9 веток. В каждом блоке 4 состояния – 3 состояния для каждой из букв алфавита, а также еще одно состояние, которое работает только тогда, когда текущие крайние 2 символа последовательности образуют симметричную пару входной. Это состояние и является допускающим, то есть “покинуть” блок не корректно(когда последнее состояние не является допускающим) невозможно. Таким образом автомат способен правильно обработать входную последовательность.

1. **Формальное описание распознавателя.**
2. **Описание в виде пятёрки множеств:**

V = {a,b,c}

K = {q0,q1,q2, …, q38,q39}

M:

M[q0,a] = q1;

M[q1,a] = q4;

M[q4,a] = q4;

M[q4,b] = q5;

M[q4,c] = q6;

M[q5,a] = q7;

M[q5,b] = q5;

M[q5,c] = q6;

M[q6,a] = q7;

M[q6,b] = q5;

M[q6,c] = q6;

M[q7,a] = q4;

M[q7,b] = q5;

M[q7,c] = q6;

M[q1,b] = q8;

M[q8,b] = q8;

M[q8,a] = q9;

M[q8,c] = q11;

M[q9,a] = q10;

M[q9,b] = q8;

M[q9,c] = q11;

M[q10,a] = q10;

M[q10,b] = q8;

M[q10,c] = q11;

M[q11,a] = q10;

M[q11,b] = q8;

M[q11,c] = q11;

M[q1,c] = q12;

M[q12,c] = q12;

M[q12,b] = q15;

M[q12,a] = q13;

M[q13,a] = q14;

M[q13,b] = q15;

M[q13,c] = q12;

M[q14,a] = q14;

M[q14,b] = q15;

M[q14,c] = q12;

M[q15,a] = q14;

M[q15,b] = q15;

M[q15,c] = q12;

M[q0,b] = q2;

M[q2,b] = q16;

M[q16,b] = q16;

M[q16,a] = q17;

M[q16,c] = q18;

M[q17,a] = q17;

M[q17,b] = q19;

M[q17,c] = q18;

M[q18,a] = q17;

M[q18,b] = q19;

M[q18,c] = q18;

M[q19,a] = q17;

M[q19,b] = q16;

M[q19,c] = q18;

M[q2,a] = q20;

M[q20,a] = q20;

M[q20,b] = q21;

M[q20,c] = q23;

M[q21,a] = q20;

M[q21,b] = q22;

M[q21,c] = q23;

M[q22,a] = q20;

M[q22,b] = q22;

M[q22,c] = q23;

M[q23,a] = q20;

M[q23,b] = q22;

M[q23,c] = q23;

M[q2,c] = q24;

M[q24,a] = q27;

M[q24,b] = q25;

M[q24,c] = q24;

M[q25,a] = q27;

M[q25,b] = q26;

M[q25,c] = q24;

M[q26,a] = q27;

M[q26,b] = q26;

M[q26,c] = q24;

M[q27,a] = q27;

M[q27,b] = q26;

M[q27,c] = q24;

M[q0,c] = q3;

M[q3,c] = q28;

M[q28,c] = q28;

M[q28,a] = q30;

M[q28,b] = q29;

M[q29,a] = q30;

M[q29,b] = q29;

M[q29,c] = q31;

M[q30,a] = q30;

M[q30,b] = q29;

M[q30,c] = q31;

M[q31,a] = q30;

M[q31,b] = q29;

M[q31,c] = q28;

M[q3,b] = q32;

M[q32,a] = q35;

M[q32,b] = q32;

M[q32,c] = q33;

M[q33,a] = q35;

M[q33,b] = q32;

M[q33,c] = q34;

M[q34,a] = q35;

M[q34,b] = q32;

M[q34,c] = q34;

M[q35,a] = q35;

M[q35,b] = q32;

M[q35,c] = q34;

M[q3,a] = q36;

M[q36,a] = q36;

M[q36,b] = q39;

M[q36,c] = q37;

M[q37,a] = q36;

M[q37,b] = q39;

M[q37,c] = q38;

M[q38,a] = q36;

M[q38,b] = q39;

M[q38,c] = q38;

M[q39,a] = q36;

M[q39,b] = q39;

M[q39,c] = q38;

S = {q0}

Z = {q4, q9, q13, q16, q21, q25, q28, q33, q37}

1. **Описание в виде таблицы переходов:**

Таблица 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **a** | **b** | **c** | **Y/N** |
| q0 | q1 | q2 | q3 | N |
| q1 | q4 | q8 | q12 | N |
| q2 | q20 | q16 | q24 | N |
| q3 | q36 | q32 | q28 | N |
| q4 | q4 | q5 | q6 | Y |
| q5 | q7 | q5 | q6 | N |
| q6 | q7 | q5 | q6 | N |
| q7 | q4 | q5 | q6 | N |
| q8 | q9 | q8 | q11 | N |
| q9 | q10 | q8 | q11 | Y |
| q10 | q10 | q8 | q11 | N |
| q11 | q10 | q8 | q11 | N |
| q12 | q13 | q15 | q12 | N |
| q13 | q14 | q15 | q12 | Y |
| q14 | q14 | q15 | q12 | N |
| q15 | q14 | q15 | q12 | N |
| q16 | q17 | q16 | q18 | Y |
| q17 | q17 | q19 | q18 | N |
| q18 | q17 | q19 | q18 | N |
| q19 | q17 | q16 | q18 | N |
| q20 | q20 | q21 | q23 | N |
| q21 | q20 | q22 | q23 | Y |
| q22 | q20 | q22 | q23 | N |
| q23 | q20 | q22 | q23 | N |
| q24 | q27 | q25 | q24 | N |
| q25 | q27 | q26 | q24 | Y |
| q26 | q27 | q26 | q24 | N |
| q27 | q27 | q26 | q24 | N |
| q28 | q30 | q29 | q28 | Y |
| q29 | q30 | q29 | q31 | N |
| q30 | q30 | q29 | q31 | N |
| q31 | q30 | q29 | q28 | N |
| q32 | q35 | q32 | q33 | N |
| q33 | q35 | q32 | q34 | Y |
| q34 | q35 | q32 | q34 | N |
| q35 | q35 | q32 | q34 | N |
| q36 | q36 | q39 | q37 | N |
| q37 | q36 | q39 | q38 | Y |
| q38 | q36 | q39 | q38 | N |
| q39 | q36 | q39 | q38 | N |

1. **Описание в виде диаграммы переходов автомата:**

Рисунок 1

**A diagram of a circuit

Description automatically generated**

1. **Набор тестов с пошаговой ручной проверкой**
2. ‘aa’ – q0(a)->q1(a)->q4 – Yes;
3. ‘aba’ – q0(a)->q1(b)->q8(a)->q9 – Yes;
4. ‘aca’ – q0(a)->q1(c)->q12(a)->q13 – Yes;
5. ‘abab’ – q0(a)->q1(b)-q8(a)->q9(b)->q8 – No;
6. ‘abba’ – q0(a)->q1(b)->q8(b)->q8(a)->q9 – Yes;
7. ‘aaaa’ – q0(a)->q1(a)->q4(a)->q4(a)->q4 – Yes;
8. ‘abcba’ – q0(a)->q1(b)->q8(c)->q11(b)->q8(a)->q9 – Yes;
9. ‘acabca’ – q0(a)->q1(c)->q12(a)->q13(b)->q15(c)->q12(a)->q13 – Yes;
10. ‘abacbbabba’ – q0(a)->q1(b)->q8(a)->q9(c)->q11(b)->q8(b)->q8(a)->q9(b)->q8(a)->q9 – Yes;
11. ‘aabcaa’ – q0(a)->q1(a)->q4(b)->q5(c)->q6(a)->q7(a)->q4 – Yes;
12. ‘acabcabca’ – q0(a)->q1(c)->q12(a)->q13(b)->q15(c)->q12(a)->q13(b)->q15(c)->q12(a)->q13 – Yes;
13. ‘abcbcaa’ – q0(a)->q1(b)->q12(c)->q12(b)->q15(c)->q12(a)->q13(a)->q14 – No;
14. ‘bb’ – q0(b)->q2(b)->q16 – Yes;
15. ‘bbb’ – q0(b)->q2(b)->q16(b)->q16 – Yes;
16. ‘bab’ – q0(b)->q2(a)->q20(b)->q21 – Yes;
17. ‘bcb’ – q0(b)->q2(c)->q24(b)->q25 – Yes;
18. ‘bbacbb’ – q0(b)->q2(b)->q16(a)->q17(c)->q18(b)->q19(b)->q16 – Yes;
19. ‘bacaab’ – q0(b)->q2(a)->q20(c)->q23(a)->q20(a)->q20(b)->q21 – Yes;
20. ‘bccabcb’ – q0(b)->q2(c)->q24(c)->q24(a)->q27(b)->q26(c)->q24(b)->q25 – Yes;
21. ‘cc’ – q0(c)->q3(c)->q28 – Yes;
22. ‘ccc’ – q0(c)->q3(c)->q28(c)->q28 – Yes;
23. ‘cac’ – q0(c)->q3(a)->q36(c)->q37 – Yes;
24. ‘cbc’ – q0(c)->q3(b)->q32(c)->q33 – Yes;
25. ‘ccabcc’ – q0(c)->q3(c)->q28(a)->q30(b)->q29(c)->q31(c)->q28 – Yes;
26. ‘caabac’ – q0(c)->q3(a)->q36(a)->q36(b)->q38(a)->q36(c)->q37 – Yes;
27. ‘cbabbc’ – q0(c)->q3(b)->q32(a)->q35(b)->q32(b)->q32(c)->q33 – Yes;
28. ‘bbabc’ – q0(b)->q2(b)->q16(a)->q17(b)->q19(c)->q18 – No;
29. ‘aaab’ – q0(a)->q1(a)->q4(a)->q4(b)->q5 – No;
30. ‘caabca’ – q0(c)->q3(a)->q36(a)->q36(b)->q39(c)->q38(a)->q36 – No;
31. ‘cb’ – q0(c)->q3(b)->q32 – No;
32. ‘ba’ – q0(b)->q2(a)->q20 – No;
33. ‘ccbc’ – q0(c)->q3(c)->q28(b)->q29(c)->q31 – No;
34. ‘acccba’ – q0(a)->q1(c)->q12(c)->q12(c)->q12(b)->q15(a)->q14 – No;
35. ‘baaacb’ – q0(b)->q2(a)->q20(a)->q20(a)->q20(c)->q23(b)->q22 – No;
36. ‘baaaab’ – q0(b)->q2(a)->q20(a)->q20(a)->q20(a)->q20(b)->q21 – Yes;
37. **Скриншоты выполнения программы на тестовых примерах**

Рисунок 2

* 1. A black screen with white text

     Description automatically generated

Рисунок 3

* 1. A black screen with white text

     Description automatically generated

Рисунок 4

* 1. A black screen with white text

     Description automatically generated

Рисунок 5

* 1. A screen shot of a black background

     Description automatically generated

Рисунок 6

* 1. A black background with green text

     Description automatically generated

Рисунок 7

* 1. A black screen with white text

     Description automatically generated

Рисунок 8

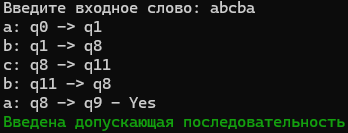
* 1. 

Рисунок 9

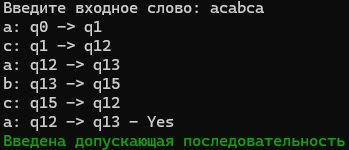
* 1. 

Рисунок 10

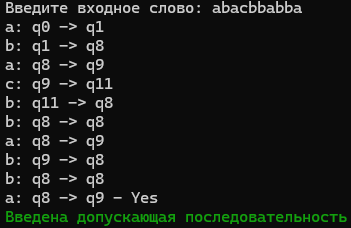
* 1. 

Рисунок 11

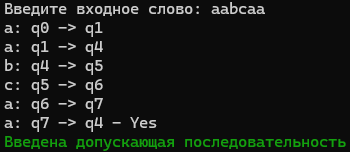
* 1. 

Рисунок 12

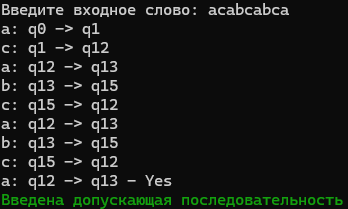
* 1. 

Рисунок 13

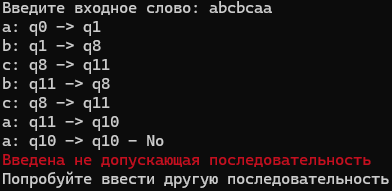
* 1. 

Рисунок 14

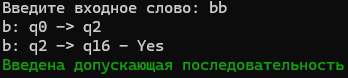
* 1. 

Рисунок 15

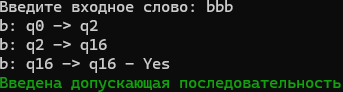
* 1. 

Рисунок 16

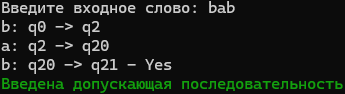
* 1. 

Рисунок 17

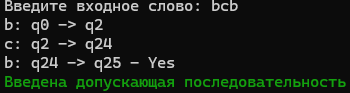
* 1. 

Рисунок 18

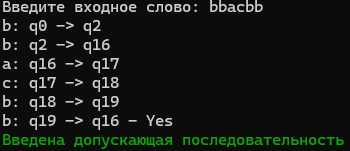
* 1. 

Рисунок 19

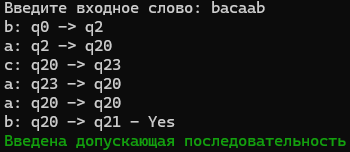
* 1. 

Рисунок 20

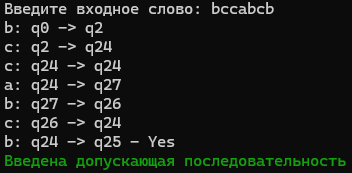
* 1. 

Рисунок 21

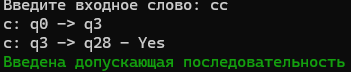
* 1. 

Рисунок 22

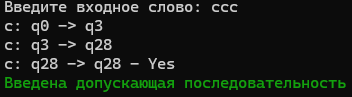
* 1. 

Рисунок 23

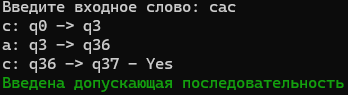
* 1. 

Рисунок 24

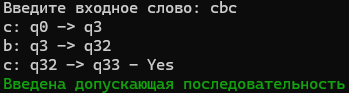
* 1. 

Рисунок 25

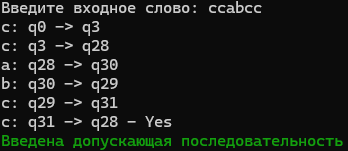
* 1. 

Рисунок 26

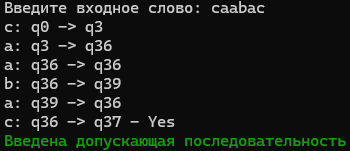
* 1. 

Рисунок 27

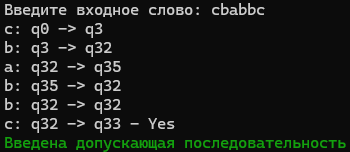
* 1. 

Рисунок 28

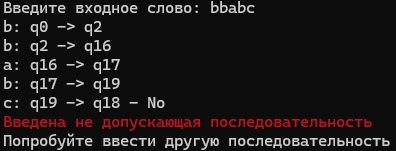
* 1. 

Рисунок 29

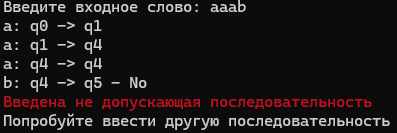
* 1. 

Рисунок 30

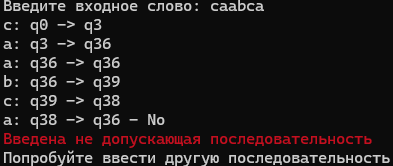
* 1. 

Рисунок 31

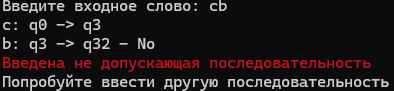
* 1. 

Рисунок 32

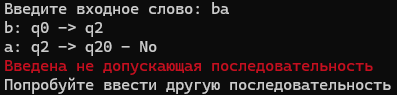
* 1. 

Рисунок 33

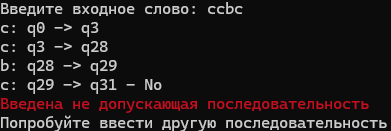
* 1. 

Рисунок 34

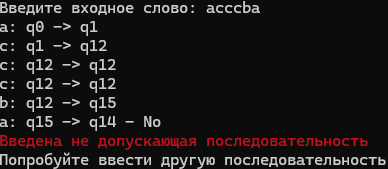
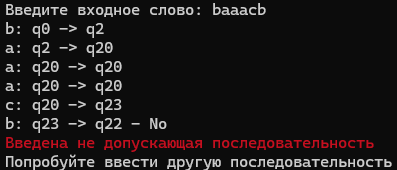
* 1. 

Рисунок 35

* 1. 

1. **Описание структуры данных, используемой в программе для хранения системы переходов автомата**

В качестве структуры данных, используемой в программе для хранения системы переходов автомата был использован контейнер “map”, первым элементом которого является “string”, являющийся состоянием автомата. Второй же элемент “map”а – также “map”. Он же в свою очередь состоит из “char” и “string”, где “string” – состояние в которое ведёт состояние, отраженное в первом элементе “map”, а “char” – символ, который относится к этому состоянию.

Неполный вид таблицы в исходном коде программы:

Рисунок 36

A black background with text and symbols

Description automatically generated

1. **Словесное описание идеи программной реализации одного шага работы автомата**

Идея программы очень проста, имеется “map”, являющийся таблицей переходов автомата, а также введеная последовательность. Есть цикл, который перебирает все символы в последовательности. Также, имеется изначальное состояние автомата – q0. На каждом шаге работы автомата, в зависимости от символа, на котором сейчас находится итератор, используя таблицу переходов, выбирается следующее состояние. Цикл заканчивается тогда, когда всё символы последовательности были обработаны, следовательно была построена полная цепочка состояний, у которой есть конечное состояние, которое затем ищется в коллекции допускающих состояний. Если оно было найдено – цепочка допускающая. Иначе – не допускающая.

1. **Листинг программы**

//ЮФУ, ИКТИБ, МОП ЭВМ

//Программирование и основы теории алгоритмов

//ДЗ1 - Проектирование и реализация конечного распознавателя.

//КТбо1-6, Кравченко Александр Андреевич

//22.03.2024

#include <iostream>

#include <map>

#include <set>

#include <string>

#include <regex>

#include <windows.h>

#include <Lmcons.h>

using namespace std;

// Функция инициализации, обработки и хранения таблицы переходов, принимает введённую строку

// Хранит в себе таблицу переходов, а также принимает введенную пользователем последовательность

// Перед отправкой введеной последовательности на проверку допускающего состояния, использует функцию sequenceValid,

// и в зависимости от возвращенного этой функцией значения либо продолжает работу над строкой, либо передает в main код ошибки,

// возвращенный функцией sequenceValid

// Если sequenceValid прошла без ошибок, проверяет строку на допускающее состояние функцией sequenceChecker

// Если вернулось значение 1, то значит последовательность допускающая, иначе возвращает 0

// Структура данных:

// Контейнер “map”, первым элементом которого является “string”,

// являющийся состоянием автомата. Второй же элемент “map”а – также “map”.

// Он же, в свою очередь, состоит из “char” и “string”, где “string” – состояние в которое ведёт состояние,

// отраженное в первом элементе “map”, а “char” – символ, который относится к этому состоянию.

int DFA(string);

// Функция, проверяющая последовательность на корректность ввода, а также проверяющая, не был ли введен символ выхода

// Возвращает значения:

// 5 - если введен символ выхода

// 2 - если введенная строка пустая

// 3 - если в последовательности используются символы не из алфавита(a,b,c)

// 4 - если длина строки меньше 2(отсутствие пары)

int sequenceValid(string);

// Главная функция проверки последовательности, принимает введённую строку, а также таблицу переходов

// Последовательно идёт по каждому символу последовательности, начиная с состояния q0

// Для каждого символа получает его следующее состояние из таблицы переходов transitionTable и сохраняет его в переменной nextState

// Пройдя по всем символам последовательности проверяет, является ли последнее состояние допускающим(ищёт его в сете допускающих состояний valids)

// Если состояние допускающее - возвращает 1

// Иначе - 0

int sequenceChecker(string, const map<string, map<char, string>>& );

int main()

{

    HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

    setlocale(LC\_ALL, "Russian");

    int returnCode;

    string input;

    SetConsoleTextAttribute(hConsole, 7);

    TCHAR username[UNLEN + 1];

    DWORD username\_len = UNLEN + 1;

    if (GetUserName(username, &username\_len)) {

        wcout << L"Здравствуйте!, " << username << endl;

    }

    while (true)

    {

        cout << "Введите входное слово или введите '0' для выхода из программы: ";

        cin>>input;

        returnCode = DFA(input);

        switch (returnCode) {

        case 0:

            cout << " - No" << endl;

            SetConsoleTextAttribute(hConsole, 4);

            cout << "Введена не допускающая последовательность" << endl;

            SetConsoleTextAttribute(hConsole, 7);

            cout << "Попробуйте ввести другую последовательность" << endl;

            break;

        case 1:

            cout << " - Yes" << endl;

            SetConsoleTextAttribute(hConsole, 2);

            cout << "Введена допускающая последовательность" << endl;

            break;

        case 2:

            SetConsoleTextAttribute(hConsole, 4);

            cout << "Введена пустая последовательность" << endl;

            SetConsoleTextAttribute(hConsole, 7);

            cout << "Попробуйте ввести другую последовательность" << endl;

            break;

        case 3:

            SetConsoleTextAttribute(hConsole, 4);

            cout << "Последовательность содержит недопустимые символы" << endl;

            SetConsoleTextAttribute(hConsole, 7);

            cout << "Попробуйте ввести другую последовательность, используя символы a,b,c" << endl;

            break;

        case 4:

            SetConsoleTextAttribute(hConsole, 4);

            cout << "Длина строки менее 2, невозможно найти пару" << endl;

            SetConsoleTextAttribute(hConsole, 7);

            cout << "Попробуйте ввести другую последовательность" << endl;

            break;

        case 5:

            cout << "Спасибо за использование!";

            exit(0);

            break;

        }

        SetConsoleTextAttribute(hConsole, 7);

        cout << endl;

    }

}

int sequenceValid(string input) {

    const regex validSymbols("^(a|b|c)+$");

    if (input == "0") return 5;

    if (input.empty()) return 2;

    if (!regex\_match(input, validSymbols)) return 3;

    if (input.length() < 2) return 4;

    return 0;

}

int sequenceChecker(string input, const map<string, map<char, string>>& transitionTable)

{

    string nextState,currentState = "q0";

    set<string> valids = { "q4","q9","q13","q16","q21","q25","q28","q33","q37" };

    for (int i = 0; i < input.size(); ++i) {

        nextState = transitionTable.at(currentState).at(input[i]);

        cout << input[i] << ": " << currentState << " -> " << nextState;

        if (i != input.size() - 1)

            cout << endl;

        currentState = nextState;

    }

    if (valids.contains(currentState)) return 1;

    else return 0;

}

int DFA(string input)

{

    map<string, map<char, string>> transitionTable =

    {

        {"q0" , { {'a', "q1" }, {'b', "q2"}, {'c', "q3"} } },

        {"q1" , { {'a', "q4" }, {'b', "q8"}, {'c', "q12"} } },

        {"q2" , { {'a', "q20" }, {'b', "q16"}, {'c', "q24"} } },

        {"q3" , { {'a', "q36" }, {'b', "q32"}, {'c', "q28"} } },

        {"q4" , { {'a', "q4" }, {'b', "q5"}, {'c', "q6"} } },

        {"q5" , { {'a', "q7" }, {'b', "q5"}, {'c', "q6"} } },

        {"q6" , { {'a', "q7" }, {'b', "q5"}, {'c', "q6"} } },

        {"q7" , { {'a', "q4" }, {'b', "q5"}, {'c', "q6"} } },

        {"q8" , { {'a', "q9" }, {'b', "q8"}, {'c', "q11"} } },

        {"q9" , { {'a', "q10" }, {'b', "q8"}, {'c', "q11"} } },

        {"q10" , { {'a', "q10" }, {'b', "q8"}, {'c', "q11"} } },

        {"q11" , { {'a', "q10" }, {'b', "q8"}, {'c', "q11"} } },

        {"q12" , { {'a', "q13" }, {'b', "q15"}, {'c', "q12"} } },

        {"q13" , { {'a', "q14" }, {'b', "q15"}, {'c', "q12"} } },

        {"q14" , { {'a', "q14" }, {'b', "q15"}, {'c', "q12"} } },

        {"q15" , { {'a', "q14" }, {'b', "q15"}, {'c', "q12"} } },

        {"q16" , { {'a', "q17" }, {'b', "q16"}, {'c', "q18"} } },

        {"q17" , { {'a', "q17" }, {'b', "q19"}, {'c', "q18"} } },

        {"q18" , { {'a', "q17" }, {'b', "q19"}, {'c', "q18"} } },

        {"q19" , { {'a', "q17" }, {'b', "q16"}, {'c', "q18"} } },

        {"q20" , { {'a', "q20" }, {'b', "q21"}, {'c', "q23"} } },

        {"q21" , { {'a', "q20" }, {'b', "q22"}, {'c', "q23"} } },

        {"q22" , { {'a', "q20" }, {'b', "q22"}, {'c', "q23"} } },

        {"q23" , { {'a', "q20" }, {'b', "q22"}, {'c', "q23"} } },

        {"q24" , { {'a', "q27" }, {'b', "q25"}, {'c', "q24"} } },

        {"q25" , { {'a', "q27" }, {'b', "q26"}, {'c', "q24"} } },

        {"q26" , { {'a', "q27" }, {'b', "q26"}, {'c', "q24"} } },

        {"q27" , { {'a', "q27" }, {'b', "q26"}, {'c', "q24"} } },

        {"q28" , { {'a', "q30" }, {'b', "q29"}, {'c', "q28"} } },

        {"q29" , { {'a', "q30" }, {'b', "q29"}, {'c', "q31"} } },

        {"q30" , { {'a', "q30" }, {'b', "q29"}, {'c', "q31"} } },

        {"q31" , { {'a', "q30" }, {'b', "q29"}, {'c', "q28"} } },

        {"q32" , { {'a', "q35" }, {'b', "q32"}, {'c', "q33"} } },

        {"q33" , { {'a', "q35" }, {'b', "q32"}, {'c', "q34"} } },

        {"q34" , { {'a', "q35" }, {'b', "q32"}, {'c', "q34"} } },

        {"q35" , { {'a', "q35" }, {'b', "q32"}, {'c', "q34"} } },

        {"q36" , { {'a', "q36" }, {'b', "q39"}, {'c', "q37"} } },

        {"q37" , { {'a', "q36" }, {'b', "q39"}, {'c', "q38"} } },

        {"q38" , { {'a', "q36" }, {'b', "q39"}, {'c', "q38"} } },

        {"q39" , { {'a', "q36" }, {'b', "q39"}, {'c', "q38"} } },

    };

    int sqVaR = sequenceValid(input);

    if (sqVaR != 0) return sqVaR;

    if (sequenceChecker(input, transitionTable)) return 1;

    return 0;

}