

Министерство образования и науки РФ  
Пермский национальный исследовательский политехнический университет  
Электротехнический факультет  
Кафедра информационных технологий и автоматизированных систем

Дискретная математика и математическая логика  
Лабораторная работа № 8

Тема: «Автомат Мура»

Выполнил: студент группы  
ИВТ-23-1Б  
Долганова Диана Евгеньевна  
Проверил: ст. пр.  
Рустамханова Г. И.

## Оглавление

<b>Цель работы</b>	<b>1</b>
<b>Задачи работы</b>	<b>2</b>
<b>Этапы выполнения</b>	<b>3</b>
<b>1. Регулярное выражение</b>	<b>4</b>
<b>2. Диаграмма Мура</b>	<b>4</b>
<b>3. Таблица переходов</b>	<b>4</b>
<b>4. Программа</b>	<b>5</b>
<b>4.1 Класс Automat</b>	<b>5</b>
<b>4.1.1 Массив переходов и карта символов</b>	<b>5</b>
<b>4.1.2 Проверка слова</b>	<b>7</b>
<b>4.2. Главная функция</b>	<b>8</b>
<b>4.2.1 Настройка локали и создание экземпляра класса</b>	<b>8</b>
<b>4.2.2 Цикл для ввода и проверки слова</b>	<b>8</b>
<b>Тестирование программы</b>	<b>9</b>
<b>Заключение</b>	<b>10</b>

## **Цель работы**

Синтезировать автомат, распознающий заданный язык. Написать программу-анализатор, которая определяет, принадлежит ли слово заданному языку.

## Задачи работы

1. Синтезировать автомат для варианта 8: формальный язык задан в алфавите  $\{a, b, c, d\}$  и содержит слова любой длины, в которых гласные и согласные буквы перемежаются.
2. Написать регулярное выражение.
3. Сделать диаграмму Мура и таблицу переходов.
4. Протестировать программу.

## Этапы выполнения

### 1. Регулярное выражение

Регулярное выражение для алфавита, в котором согласные и гласные буквы перемежаются, выглядит следующим образом:

$$(a \cdot (b+c+d)) \cdot (b+c+d)^*$$

$$((b+c+d) \cdot (b+c+d) \cdot a)^*$$

### 2. Диаграмма Мура

Автомат Мура (рис. 1) включает следующие состояния:

q\_0 - гласные и согласные перемежаются.

q\_1 - последний символ - гласная (a).

q\_2 - последний символ - согласная (b, c, d).

q\_3 - ошибка (последние два символа одного типа).

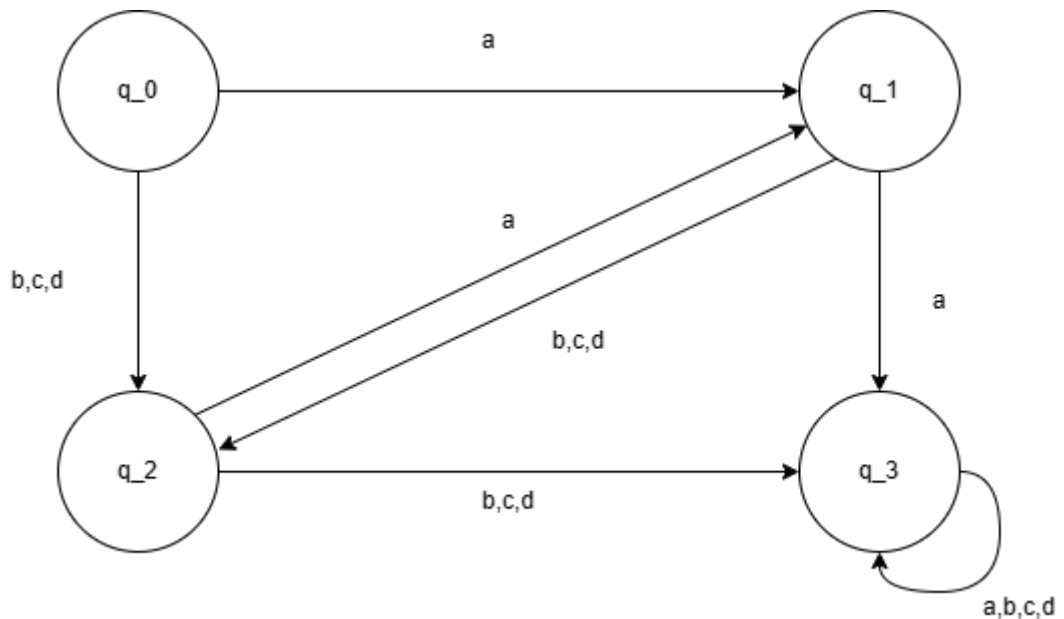


Рисунок 1 - Диаграмма Мура

### 3. Таблица переходов

Таблица переходов выглядит следующим образом:

Таблица 1 - Таблица переходов

Текущее состояние	Вход: a	Вход: b,c,d
q_0	q_1	q_2
q_1	q_3	q_2
q_2	q_1	q_3
q_3	q_3	q_3

## 4. Программа

Программа была разработана на языке C++ в программе среде Microsoft Visual Studio 2022.

### 4.1 Класс Automaton

Объявляем класс Automaton, который представляет автомат для проверки строк в заданном языке (рис. 2).

```

    > #include <iostream>
    > #include <string>
    > #include <unordered_map>

    using namespace std;

    > class Automaton {
    >

```

Рисунок 2 - Объявление класса

#### 4.1.1 Массив переходов и карта символов

Массив *transitions* определяет возможные переходы между состояниями. Он имеет размерность 4x4, где строки представляют текущее состояние, а столбцы – входные символы:

Строки:

- 0 - начальное состояние
- 1 - последняя буква была гласной (a)
- 2 - последняя буква была согласной (b, c, d)
- 3 - состояние ошибки

Переходы:

- Из состояния 0:
  - при вводе 'a' переходит в состояние 1
  - при вводе 'b', 'c', 'd' – в состояние 2
- Из состояния 1:
  - при вводе 'a' – в состояние ошибки (3)
  - при вводе 'b', 'c', 'd' – в состояние 2
- Из состояния 2:
  - при вводе 'a' – в состояние 1
  - при вводе 'b', 'c', 'd' – в состояние ошибки (3)
- Из состояния 3 – нет законных переходов, остается в состоянии ошибки.

Карта *charToIndex* связывает символы ('a', 'b', 'c', 'd') с их индексами, которые используются для доступа к массиву переходов. Это позволяет быстро находить, какой индекс соответствует вводу (рис. 3).

```
int transitions[4][4] = {
    {1, 2, 2, 2}, // Из состояния 0: a->1, b/c/d->2
    {3, 2, 2, 2}, // Из состояния 1: a->3 (ошибка), b/c/d->2
    {1, 3, 3, 3}, // Из состояния 2: a->1, b/c/d->3 (ошибка)
    {3, 3, 3, 3}  // Состояние 3
};

unordered_map<char, int> charToIndex = {
    {'a', 0},
    {'b', 1},
    {'c', 2},
    {'d', 3}
};
```

Рисунок 3 - Массив переходов и карта символов

### 4.1.2 Проверка слова

Объявляем метод класса *isAccepted* (рис. 4), который принимает строку *word* и возвращает булево значение, указывающее, принадлежит ли слово языку:

1. Переменная *state* инициализируется нулем, что соответствует начальному состоянию автомата.
2. Цикл перебирает каждый символ строки *word*.
3. Приводим символ *ch* к нижнему регистру.
4. Проверяем, существует ли символ *ch* в *charToIndex* (допустимый ли он). Если его нет, выводит сообщение об ошибке и возвращает *false*.
5. Определяем индекс символа из *charToIndex* и обновляем состояние *state* на основе переходов из текущего состояния. Если автомат вошел в состояние ошибки (3), сразу возвращается *false*.
6. После обработки всех символов возвращаем *true*, если автомат завершил работу в одном из состояний 0, 1 или 2 (принимается), и *false*, если в состоянии 3 (ошибка).

```
public:
    bool isAccepted(const string& word) {
        int state = 0; // Начальное состояние

        for (char ch : word) {

            ch = tolower(ch);

            // Проверяем допустимость символа
            if (charToIndex.find(ch) == charToIndex.end()) {
                cout << "Недопустимый символ: " << ch << endl;
                return false;
            }

            int symbolIndex = charToIndex[ch];
            state = transitions[state][symbolIndex];

            // Если попали в состояние ошибки - возвращаем false
            if (state == 3) {
                return false;
            }
        }

        // Слово принимается, если закончили в состоянии 0, 1 или 2
        return state != 3;
    }
};
```

Рисунок 4 - Метод *isAccepted*



## 4.2. Главная функция

### 4.2.1 Настройка локали и создание экземпляра класса

Добавляем строчку для корректного отображения русского языка создаем экземпляр класса `Automat` (рис. 5).

```
int main() {  
    setlocale(LC_ALL, "RU");  
    Automat automat;  
}
```

Рисунок 5 - Локаль и создание экземпляра класса

### 4.2.2 Цикл для ввода и проверки слова

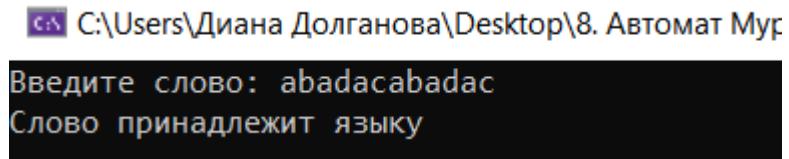
Запускаем бесконечный цикл, который запрашивает у пользователя ввести слово и сохраняет его в переменную `input`. Проверяем, принадлежит ли введенное слово языку, с помощью метода `isAccepted`. Выводим результат проверки – принадлежит ли слово языку или нет, после чего добавляем пустую строку для оформления (рис. 6).

```
while (true) {  
    cout << "Введите слово: ";  
    string input;  
    cin >> input;  
  
    bool result = automat.isAccepted(input);  
    cout << (result ? "Слово принадлежит языку" : "Слово НЕ принадлежит языку") << endl;  
    cout << endl;  
}  
  
return 0;  
}
```

Рисунок 6 - Цикл для ввода и проверки слова

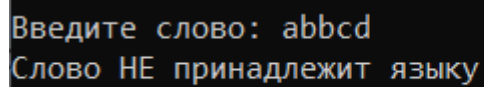
## Тестирование программы

Для проверки программы на корректность введем слово, принадлежащее языку (рис. 7) и слово, не принадлежащее языку (рис. 8).



C:\Users\Диана Долганова\Desktop\8. Автомат Мур  
Введите слово: abadacabadac  
Слово принадлежит языку

Рисунок 7 - Ввод слова, принадлежащего языку



Введите слово: abbcd  
Слово НЕ принадлежит языку

Рисунок 8 - Ввод слова, не принадлежащего языку

## **Заключение**

Была разработана программа для определения принадлежности слова заданному языку. Кроме того, были созданы диаграмма Мура и таблица переходов.

## **Список используемой литературы**

1. Автоматы Мура и Мили — Викиконспекты URL:  
[https://neerc.ifmo.ru/wiki/index.php?title=Автоматы\\_Мура\\_и\\_Мили](https://neerc.ifmo.ru/wiki/index.php?title=Автоматы_Мура_и_Мили) (дата обращения: 25.04.2025).
2. §3.2. Диаграмма Мура и таблица автомата URL:  
<https://studfile.net/preview/300184/page:2/> (дата обращения: 26.04.2025).