МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ: _____

ПРЕПОДАВАТЕЛЬ:

_____ / _____ / _____ / _____
(должность, учёная степень, звание) (подпись) (дата защиты) (инициалы, фамилия)

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

«Синтез конечных автоматов»

ПО КУРСУ: « Теория вычислительных процессов»

РАБОТУ ВЫПОЛНИЛА СТУДЕНТКА: _____ / А. Предко
(номер группы) (инициалы, фамилия)

/ _____ / _____
(подпись студента) (дата отчета)

Санкт-Петербург 2023

1. Цель работы:

   Изучить алгоритм синтеза конечных автоматов

2. Постановка задачи:

   2) (c|d)<(b|c)|d>f

   - Построить конечный автомат, который осуществляет проверку входного слова на допустимость в заданном регулярном выражении используя алгоритм синтеза конечных автоматов;
   - Привести в отчете процесс синтеза конечного автомата
   - Создать программу на языке высокого уровня реализующую алгоритм синтеза конечного автомата на основе заданного регулярного выражения;
   - Задать построенный КНА, тремя способами.

3. Конечный автомат заданный тремя способами:

   (c|d)<(b|c)|d>f

   S = < X, Q, U, δ, λ>

   X = {c, d, b, f}

   Q = {q0, q1, q2}

   U = {0, 1}

   1) Матрица переходов задает функцию отображения переходов:

      $q(i) = \delta\,(x(i), q(i-1))$

      Входная матрица:
      $\Delta[m,n]$ =

Q\X	c	d	b	f
q0	q1	q2	—	—
q1	q4	q5	q3	q6
q2	q4	q5	q3	q6
q3	q4	q5	q3	q6
q4	q4	q5	q3	q6
q5	q5	q3	q6	q4
q6	—	—	—	—

Выходная матрица:

$\Delta[m,n]$ =

Q\X	c	d	b	f
q0	0	0	—	—
q1	0	0	0	1
q2	0	0	0	1
q3	0	0	0	1
q4	0	0	0	1
q5	0	0	0	1
q6	—	—	—	—

2) Ориентированный граф(мультиграф) – Граф переходов или диаграмма переходов

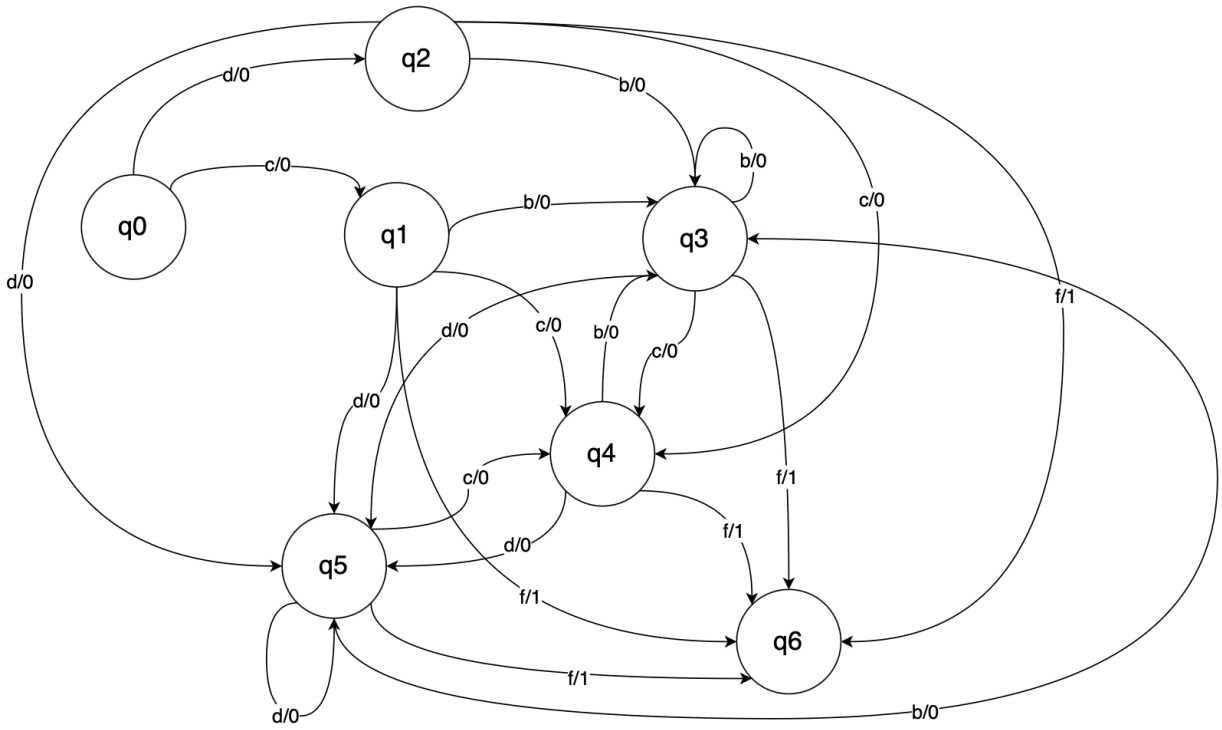

Рис. 1 - Диаграмма переходов

4. Процесс синтеза конечного автомата:

В регулярном выражении места размечают вертикальными линиями и основные места нумеруют натуральными числами. На рис. 5.9 показана разметка основных мест в выражении

4. Алгоритм синтеза конечных автоматов:

Для примера рассмотрим процесс синтеза КНА на регулярном выражении: (c|d)<(b|c)|d>f, для удобства знак «|» возьмем за «+», тогда РВ будет выглядеть следующим образом: (c+d)<(b+c)+d>f

1) В регулярном выражении места размечают вертикальными линиями и основные места нумеруют натуральными числами.

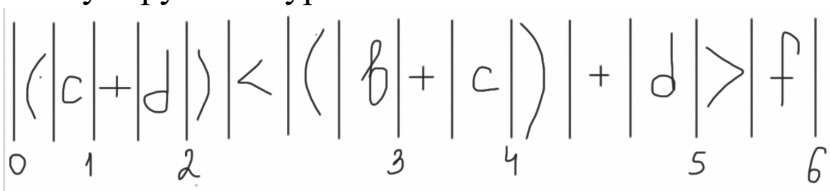

Рис. 1. — Разметка начального места и основных мест

2) Номер каждого основного места a присваивают всем неосновным местам, подчиненным месту a. Правила подчинения мест применяют в следующем порядке: 1, 2, 3, 4; правило 5 применяют одновременно с каждым из первых четырех правил; затем повторяют применение указанных правил до тех пор, пока разметка мест после его выполнения не изменяется

Правила подчинения мест в регулярных выражениях:

1. Начальные места всех слагаемых многочлена, помещенного в обычные или итерационные скобки, подчинены месту, находящемуся непосредственно слева от открывающейся скобки.

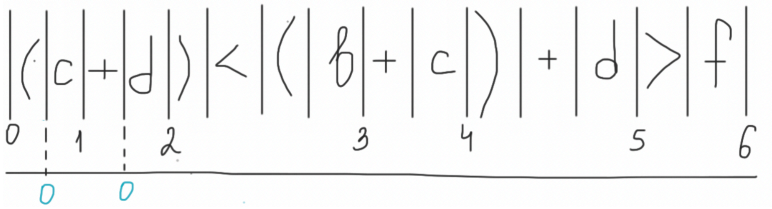

Рис. 2.1 — Применение первого правила подчинения мест в регулярном выражении

```
( c + d ) < ( b + c ) + d > f
0 0 1 0 2 . . . 3 . 4 . . 5 . 6
Program ended with exit code: 0
```

Рис. 2.2 — Программная реализация первого правила подчинения мест в регулярном выражении

2. Место, расположенное непосредственно справа от закрывающей скобки (обычной или итерационной), подчинено конечным местам всех слагаемых многочлена, заключенного в эти скобки

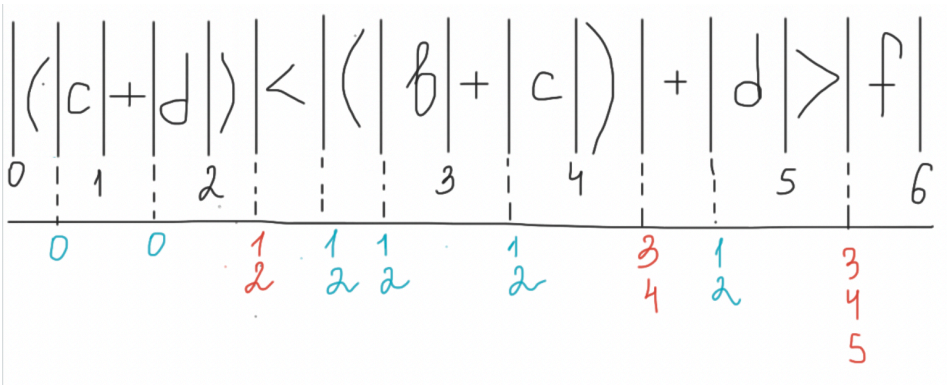

Рис. 3.1 — Применение второго правила подчинения мест в регулярном выражении

```
   (  c  +  d  )  <  (  b  +  c  )  +  d  >  f
   0  0  1  0  2  1  1  1  3  1  4  3  1  5  3  6
   .  .  .  .  .  2  2  2  .  2  .  4  2  .  4  .
   .  .  .  .  .  .  .  .  .  .  .  .  .  .  5  .
Program ended with exit code: 0
```

Рис. 3.2 — Программная реализация второго правила подчинения мест в регулярном выражении

3. Место, расположенное непосредственно справа от закрывающей итерационной скобки, подчинено месту, расположенному непосредственно слева от соответствующей открывающей итерационной скобки

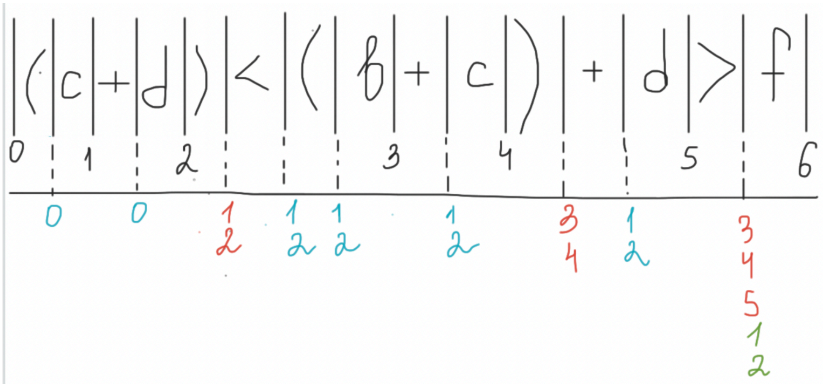

Рис. 4.1 — Применение третьего правила подчинения мест в регулярном выражении

```
   (   c   +   d   )   <   (   b   +   c   )   +   d   >   f
   0   0   1   0   2   1   1   1   3   1   4   3   1   5   1   6
   .   .   .   .   .   2   2   2   .   2   .   4   2   .   2   .
   .   .   .   .   .   .   .   .   .   .   .   .   .   .   3   .
   .   .   .   .   .   .   .   .   .   .   .   .   .   .   4   .
   .   .   .   .   .   .   .   .   .   .   .   .   .   .   5   .
Program ended with exit code: 0
```

Рис. 4.2 — Программная реализация третьего правила подчинения мест в регулярном выражении

4. Начальные места всех слагаемых многочлена, заключенного в итерационные скобки, подчинены месту, расположенному непосредственно справа от закрывающей скобки

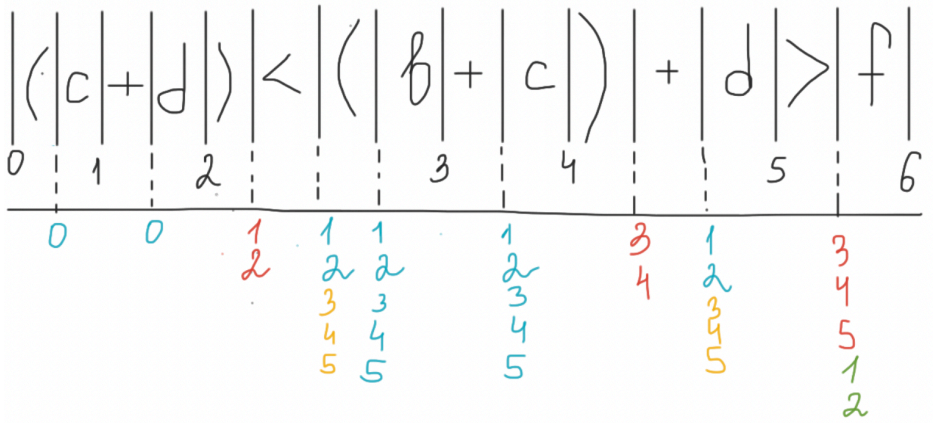

Рис. 5.1 — Применение четвёртого правила подчинения мест в регулярном выражении

```
   (   c   +   d   )   <   (   b   +   c   )   +   d   >   f
   0   0   1   0   2   1   1   1   3   1   4   3   1   5   1   6
   .   .   .   .   .   2   2   2   .   2   .   4   2   .   2   .
   .   .   .   .   .   .   3   3   .   3   .   .   3   .   3   .
   .   .   .   .   .   .   4   4   .   4   .   .   4   .   4   .
   .   .   .   .   .   .   5   5   .   5   .   .   5   .   5   .
```

Рис. 5.2 — Программная реализация четвёртого правила подчинения мест в регулярном выражении

5. Если место y подчинено месту p, а место p подчинено месту a, то место y подчинено месту a
6. Никаких других правил подчиненности мест, кроме указанных в п. 1—5, не существует

3) Строят таблицу переходов (ТП) автомата Мура по следующим правилам:

а) в качестве состояний автомата берутся подмножества множества номеров основных мест;

б) в качестве начального состояния используется начальное состояние 0 и со строки, соответствующей этому состоянию, начинается построение ТП;

в) строки, соответствующие остальным состояниям, вводятся в ТП лишь после того, как обозначающие их состояния уже появились в ранее записанных строках;

г) в клетке на пересечении столбца *cij* и строки $s_{/3}$ которой соответствует некоторое множество номеров основных мест, записывается состояние (множество номеров основных мест), состоящее из номеров всех тех и только тех основных мест, в которые возможен переход через букву *cij* из мест, в множестве номеров которых находится хотя бы один номер, принадлежащий состоянию s_r Если таких основных мест не существует, в клетке записывается знак безразличного состояния

д) в столбце «Выход» проставляется цифра 1 в строках, соответствующих отмеченным состояниям; отмеченными являются состояния, среди номеров которых находится хотя бы один номер, принадлежащий конечному месту;

е) процесс построения ТП завершается, когда всем состояниям (множествам основных мест), записанным в клетках, будет соответствовать одна из строк ТП.

Отметим предосновные места:

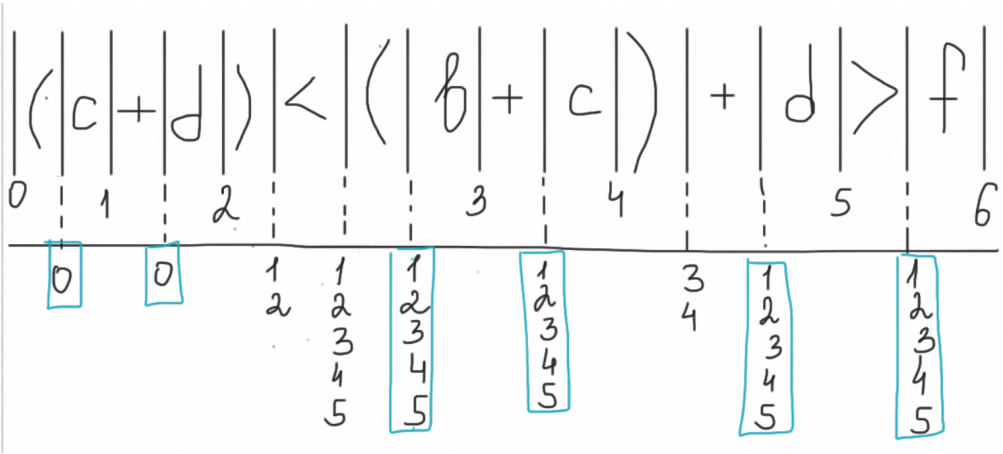

Рис. 6 — Выделение предосновных мест

Начальное состояние КНА отмечается символом пустого множества (или ноль) затем среди предосновных мест ищем у кого из них есть 0 и в случае нахождения, записываем в ячейку основное место внешнего алфавита.

После заполнения первого столбца значения ячеек переписываем отдельно в

первую строчку таблицы (шапку таблицы) и продолжаем до тех пор, пока есть новые значения, которые можно вписать в шапку таблицы

Таблица переходов:

X\Q	0	1	2	3	4	5	6
c	1/0	4/0	4/0	4/0	4/0	4/0	—
d	2/0	5/0	5/0	5/0	5/0	5/0	—
b	—	3/0	3/0	3/0	3/0	3/0	—
f	—	6/1	6/1	6/1	6/1	6/1	—

Рис. 6 — Матрица переходов

Автоматная матрица:

Q\Q	0	1	2	3	4	5	6
0	—	c/0	d/0	—	—	—	—
1	—	—	—	b/0	c/0	d/0	f/1
2	—	—	—	b/0	c/0	d/0	f/1
3	—	—	—	b/0	c/0	d/0	f/1
4	—	—	—	b/0	c/0	d/0	f/1
5	—	—	—	b/0	c/0	d/0	f/1
6	—	—	—	—	—	—	—

Рис. 7 — Автоматная матрица

5. Текстовые файлы входных и выходных данных программы:

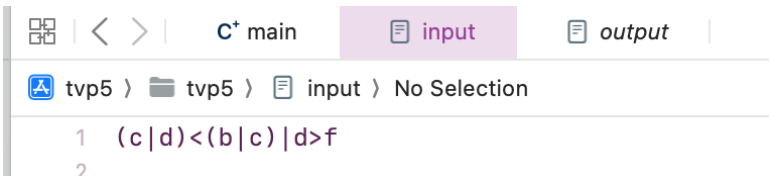

Рис. 8 — Текстовый файл входных данных

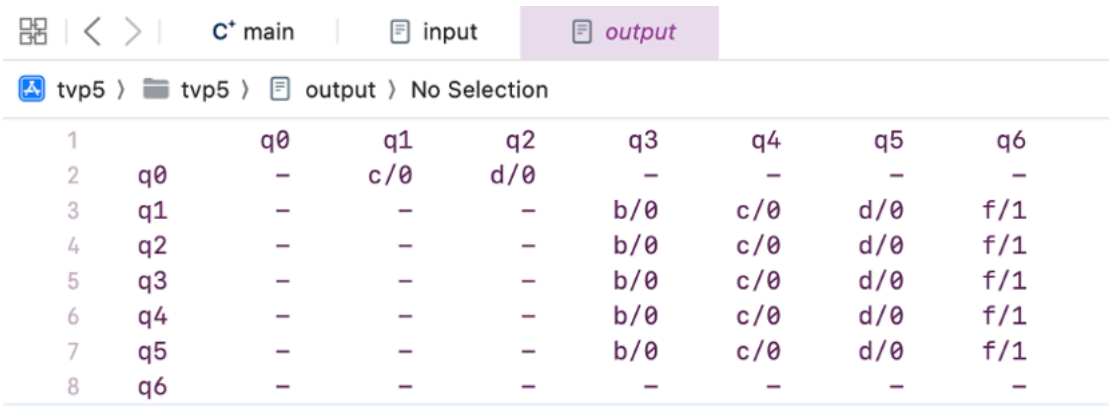

Рис. 9 — Текстовый файл выходных данных

6. Вывод

   В ходе выполнения данной работы был изучен алгоритм синтеза конечных автоматов, правила подчинения мест в регулярных выражениях, а также запрограммирован алгоритм на языке C++

Приложение 1

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
#include <set>

using namespace std;

struct MyStruct
{
    char alpha;
    int main_place;
    vector<int> ex_places;

    MyStruct(char a, int m, vector<int> e)
        : alpha(a),  main_place(m), ex_places(e) {}
};

struct FromTo {
    vector<int> from;
    vector<int> to;
    char ch;
    bool isFinal;

    FromTo(const vector<int> f, const vector<int> t, char c)
        : from(f), to(t), ch(c) {}

    FromTo(const vector<int> f)
        : from(f), ch('\0') {}
};

string get_expression() {
    ifstream in("/Users/aliona/Desktop/University
☠/4.1🚬/ТВП/tvp5/input.txt"); // Открываем файл для чтения
    if (in.is_open()) {
        string line;
        if (getline(in, line)) {
            return "(" + line + ")";
        }
        else {
            cout << "Файл пуст." << endl;
        }
    }
    in.close(); // Закрываем файл после чтения

    return 0;
}

vector<vector<int>> put_start_places(string expression) {
```

```
    vector<vector<int>> places(expression.size() + 1);

    places[0].push_back(0); // добавление начального места

    int count = 1; // счетчик количества основных мест

    for (int i = 0; i < expression.size(); i++) {
        if (isalpha(expression[i])) { // если символ - буква
            places[i + 1].push_back(count); // добавление
основного места
            count += 1;
        }
        if (expression[i] == ',') {
            places[i + 1].push_back(0); // добавление основного
места
        }
    }

    return places;
}

string get_lubstr(string expression, char bkt) {

    string result;
    int open_count = 0;
    int close_count = 0;

    for (auto& symbol : expression) {
        if (bkt == '(') {
            if (symbol == '(') {
                open_count += 1;
            }
            if (symbol == ')') {
                close_count += 1;
            }
        }
        else if (bkt == '<') {
            if (symbol == '<') {
                open_count += 1;
            }
            if (symbol == '>') {
                close_count += 1;
            }
        }
        result += symbol;
        if (open_count == close_count) {
            return result;
        }
    }
    return 0;
}
```

```
int first_rule(string expression, vector<vector<int>>& places,
int first_pos = 0) {
    places[first_pos + 1].insert(places[first_pos + 1].end(),
places[first_pos + 0].begin(), places[first_pos + 0].end());
    sort(places[first_pos + 1].begin(), places[first_pos +
1].end());  // Сначала отсортируем вектор
    places[first_pos + 1].erase(unique(places[first_pos +
1].begin(), places[first_pos + 1].end()), places[first_pos +
1].end());  // Затем удалим дубликаты
    for (int i = 1; i < expression.size(); i++) {
        if (expression[i] == '(') { // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += first_rule(sub_exp, places, i + first_pos) - 1;
        }
        if (expression[i] == '<') {  // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += first_rule(sub_exp, places, i + first_pos) - 1;
        }
        if (expression[i] == '|') {
            places[i + first_pos + 1].insert(places[i +
first_pos + 1].end(), places[first_pos + 0].begin(),
places[first_pos + 0].end());
            sort(places[i + first_pos + 1].begin(), places[i +
first_pos + 1].end());  // Сначала отсортируем вектор
            places[i + first_pos + 1].erase(unique(places[i +
first_pos + 1].begin(), places[i + first_pos + 1].end()),
places[i + first_pos + 1].end());  // Затем удалим дубликаты
        }
    }
    return expression.size();
}

int second_rule(string expression, vector<vector<int>>& places,
int first_pos = 0) {

    for (int i = 1; i < expression.size(); i++) {
        if (expression[i] == '(') { // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += second_rule(sub_exp, places, i + first_pos) -
1;
        }
        if (expression[i] == '<') {  // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += second_rule(sub_exp, places, i + first_pos) -
1;
```

```
        }
        if (expression[i] == '|') {
            places[expression.size() +
first_pos].insert(places[expression.size() + first_pos].end(),
places[i + first_pos].begin(), places[i + first_pos].end());
            sort(places[expression.size() + first_pos].begin(),
places[expression.size() + first_pos].end());  // Сначала
отсортируем вектор
            places[expression.size() +
first_pos].erase(unique(places[expression.size() +
first_pos].begin(), places[expression.size() +
first_pos].end()), places[expression.size() + first_pos].end());
// Затем удалим дубликаты
        }
    }
    places[expression.size() +
first_pos].insert(places[expression.size() + first_pos].end(),
places[expression.size() + first_pos - 1].begin(),
places[expression.size() + first_pos - 1].end());
    sort(places[expression.size() + first_pos].begin(),
places[expression.size() + first_pos].end());  // Сначала
отсортируем вектор
    places[expression.size() +
first_pos].erase(unique(places[expression.size() +
first_pos].begin(), places[expression.size() +
first_pos].end()), places[expression.size() + first_pos].end());
// Затем удалим дубликаты

    return expression.size();
}

int third_rule(string expression, vector<vector<int>>& places,
int first_pos = 0) {

    for (int i = 1; i < expression.size(); i++) {
        if (expression[i] == '<') {  // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += third_rule(sub_exp, places, i + first_pos) - 1;

        }
    }
    if (expression[0] == '<') {
        places[expression.size() +
first_pos].insert(places[expression.size() + first_pos].end(),
places[first_pos].begin(), places[first_pos].end());
        sort(places[expression.size() + first_pos].begin(),
places[expression.size() + first_pos].end());  // Сначала
отсортируем вектор
        places[expression.size() +
first_pos].erase(unique(places[expression.size() +
first_pos].begin(), places[expression.size() +
```

```
first_pos].end()), places[expression.size() + first_pos].end());
// Затем удалим дубликаты
    }

    return expression.size();
}

int fourth_rule(string expression, vector<vector<int>>& places,
int first_pos = 0) {

    for (int i = 1; i < expression.size(); i++) {
        if (expression[i] == '<') {  // если нашли открывающую
скобку
            string sub_exp = get_lubstr(expression.substr(i,
expression.size() - i), expression[i]);
            i += fourth_rule(sub_exp, places, i + first_pos) -
1;
        }
        if (expression[0] == '<' && expression[i] == '|') {
            places[first_pos + i + 1].insert(places[first_pos +
i + 1].end(), places[expression.size() + first_pos].begin(),
places[expression.size() + first_pos].end());
            sort(places[first_pos + i + 1].begin(),
places[first_pos + i + 1].end());  // Сначала отсортируем вектор
            places[first_pos + i +
1].erase(unique(places[first_pos + i + 1].begin(),
places[first_pos + i + 1].end()), places[first_pos + i +
1].end());  // Затем удалим дубликаты
        }
    }
    if (expression[0] == '<') {
        places[first_pos + 1].insert(places[first_pos +
1].end(), places[expression.size() + first_pos].begin(),
places[expression.size() + first_pos].end());
        sort(places[first_pos + 1].begin(), places[first_pos +
1].end());  // Сначала отсортируем вектор
        places[first_pos + 1].erase(unique(places[first_pos +
1].begin(), places[first_pos + 1].end()), places[first_pos +
1].end());  // Затем удалим дубликаты
    }

    return expression.size();
}


vector<MyStruct> get_alphas_with_places(string expression,
vector<vector<int>> places) {
    vector<MyStruct> alphas_with_places;
    int main_place = 1;

    for (int i = 0; i < expression.size(); i++) {
        if (isalpha(expression[i])) {
```

```
            alphas_with_places.push_back(MyStruct(expression[i],
main_place, places[i]));
            main_place += 1;
        }
    }

    return alphas_with_places;
}

vector<int> get_alpha_main_place(vector<MyStruct>
alphas_with_places, vector<int> q, char x) {
    vector<int> main_places;
    for (auto elem : alphas_with_places) {
        if (elem.alpha == x) {
            for (int i = 0; i < q.size(); i++) {
                if (find(elem.ex_places.begin(),
elem.ex_places.end(), q[i]) != elem.ex_places.end()) {
                    main_places.push_back(elem.main_place);
                }
            }
        }
    }
    if (main_places.size())
        return main_places;
    else
        return { -1 };
}

vector<vector<vector<int>>> get_table(vector<MyStruct>
alphas_with_places, string expression) {
    vector<char> rows;
    for (char c : expression) {
        if (isalpha(c) && find(rows.begin(), rows.end(), c) ==
rows.end()) {
            rows.push_back(c);
        }
    }

    vector<vector<vector<int>>> table(rows.size() + 1);
    table[0].push_back({ 0 });

    // заполняет таблицу по столбцам
    for (int j = 0; j < table[0].size(); j++) // итератор по
столбцам
    {
        for (int i = 1; i < table.size(); i++) // итератор по
строкам
        {
            vector<int> founded_q =
get_alpha_main_place(alphas_with_places, table[0][j], rows[i -
1]);
            sort(founded_q.begin(), founded_q.end());  //
Сначала отсортируем вектор
```

```
            founded_q.erase(unique(founded_q.begin(),
founded_q.end()), founded_q.end());  // Затем удалим дубликаты

            table[i].push_back(founded_q);

            if (founded_q[0] != -1) {
                if (find(table[0].begin(), table[0].end(),
founded_q) == table[0].end()) {
                    table[0].push_back(founded_q);
                }
            }
        }
    }
    return table;
}

void print_table(const vector<vector<vector<int>>>& table,
string expression) {
    vector<char> rows;
    rows.push_back(' ');
    set<char> uniqueChars;
    for (char c : expression) {
        if (isalpha(c) && uniqueChars.find(c) ==
uniqueChars.end()) {
            rows.push_back(c);
            uniqueChars.insert(c);
        }
    }
    // Вывод значений вектора rows
//    cout << "Unique letters in expression: ";
//    for (char letter : rows) {
//        cout << letter << " ";
//    }
//    cout << endl;

    int count = 0;

    for (const auto& row : table) {
        cout << rows[count] << "\t";
        for (const auto& column : row) {
            for (const auto& element : column) {
                if (element != -1)
                    cout << element << " ";
                else
                    cout << "-";
            }
            cout << "\t";
        }
        count += 1;
        cout << endl;
    }
}
```

```
vector<FromTo> get_from_to_vec(vector<vector<vector<int>>>
table, vector<char> rows) {
    vector<FromTo> result;

    for (int j = 0; j < table[0].size(); j++) {
        vector<int> from = (table[0][j]);
        char ch;
        for (int i = 1; i < table.size(); i++) {
            if (table[i][j][0] != -1) {
                vector<int> to = table[i][j];
                ch = rows[i - 1];
                result.push_back(FromTo(from, to, ch));
            }
        }
    }

    return result;
}

void print_matrix(const vector<vector<vector<int>>>& table,
string expression, vector<vector<int>> places) {

    ofstream outputFile("/Users/aliona/Desktop/University
☠️/4.1🚬/ТВП/tvp5/output.txt", ios::app);

    vector<char> rows;
    for (char c : expression) {
        if (isalpha(c) && find(rows.begin(), rows.end(), c) ==
rows.end()) {
            rows.push_back(c);
        }
    }

    vector<int> last_place = places.back();

    for (auto const& a:last_place){
        cout<<a<<" ";
    }
    cout<<endl;

    vector<FromTo> ftv = get_from_to_vec(table, rows);

    vector<vector<string>> matrix(table[0].size() + 1,
vector<string>(table[0].size() + 1, "-"));

    for (const auto& i : ftv) {
        char data = i.ch;
        string suffix = "/0";

        for (auto j : i.from) {
            for (auto n : i.to) {
                if (find(last_place.begin(), last_place.end(),
n) != last_place.end()) {
```

```
                    suffix = "/1";
                }
                matrix[j + 1][n + 1] = data + suffix;
                suffix = "/0"; // Сбросим суффикс обратно на /0
для следующей итерации
            }
        }
    }

    matrix[0][0] = "";
    for (int i = 1; i < matrix[0].size(); i++) {
        matrix[0][i] = "q" + to_string(i - 1);
        matrix[i][0] = "q" + to_string(i - 1);
    }
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix.size(); j++) {
            outputFile << setw(4)<< matrix[i][j] << "\t";
            cout << setw(4)<< matrix[i][j] << "\t";
        }
        outputFile << endl;
        cout << endl;
    }

    return;
}

void printPlaces(vector<vector<int>> places, string expression){

    // Найти максимальное количество элементов в столбце
    size_t maxColumnSize = 0;
    for (const auto& innerVec : places) {
        maxColumnSize = max(maxColumnSize, innerVec.size());
    }

    cout << setw(2)<<" ";
    for(int i =0; i<expression.size();i++){
        cout << setw(4)<<expression[i];
    }
    cout<<endl;
    // Вывод элементов вертикально с выравниванием по столбцам
    for (size_t row = 0; row < maxColumnSize; ++row) {
        for (const auto& innerVec : places) {
            if (row < innerVec.size()) {
                cout << setw(4) << innerVec[row];
            } else {
                cout << setw(4) << ".";
            }
        }
        cout << endl;
    }
}

int main()
```

```
{
    string expression = get_expression();

    vector<vector<int>> places = put_start_places(expression);
    vector<vector<int>> ex_places;

    while (places != ex_places) {
        ex_places = places;
        first_rule(expression, places);
        second_rule(expression, places);
        third_rule(expression, places);
        fourth_rule(expression, places);
    }

    places.pop_back();
    places.erase(places.begin());
    expression.pop_back();
    expression.erase(expression.begin());


    vector<MyStruct> alphas_with_places =
get_alphas_with_places(expression, places);

//    // Вывод значений после добавления в конец
//    for (const auto& item : alphas_with_places) {
//        cout << "Alpha: " << item.alpha << ", Main Place: " <<
item.main_place << ", Extra Places: ";
//        for (int place : item.ex_places) {
//            cout << place << " ";
//        }
//        cout << endl;
//    }




    vector<char> rows;
    rows.push_back('a');
    rows.push_back('b');

    vector<vector<vector<int>>> table =
get_table(get_alphas_with_places(expression, places),
expression);

    vector<FromTo> result = get_from_to_vec(table, rows);

//    // Вывод содержимого вектора result
//    for (const FromTo& element : result) {
//        // Здесь используйте методы доступа для вывода
содержимого элемента FromTo
//        cout << "From: ";
//        for (int fromVal : element.from) {
```

```
//             cout << fromVal << " ";
//         }
//         cout << "| To: ";
//         for (int toVal : element.to) {
//             cout << toVal << " ";
//         }
//         cout << "| Ch: " << element.ch << endl;
//     }


    printPlaces(places,expression);


    cout<<endl;
    cout <<"Входная матрица" <<endl;
    print_table(table, expression);
    cout << endl;

    ofstream outputFile("/Users/aliona/Desktop/University
☠️/4.1🚬/ТВП/tvp5/output.txt", ios::trunc);

    print_matrix(table, expression, places);



    return 0;
}
```