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

1. Изображение выглядит как Шрифт, типография, каллиграфия, белый

   Автоматически созданное описаниеПостановка задачи:

•  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.

1. 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*) = δ (*x*(*i*),*q*(*i* −1))

Входная матрица:  
Δ[*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:

Δ[*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 | — | — | — | — |

1. Ориентированный граф(мультиграф) – Граф переходов или диаграмма переходов

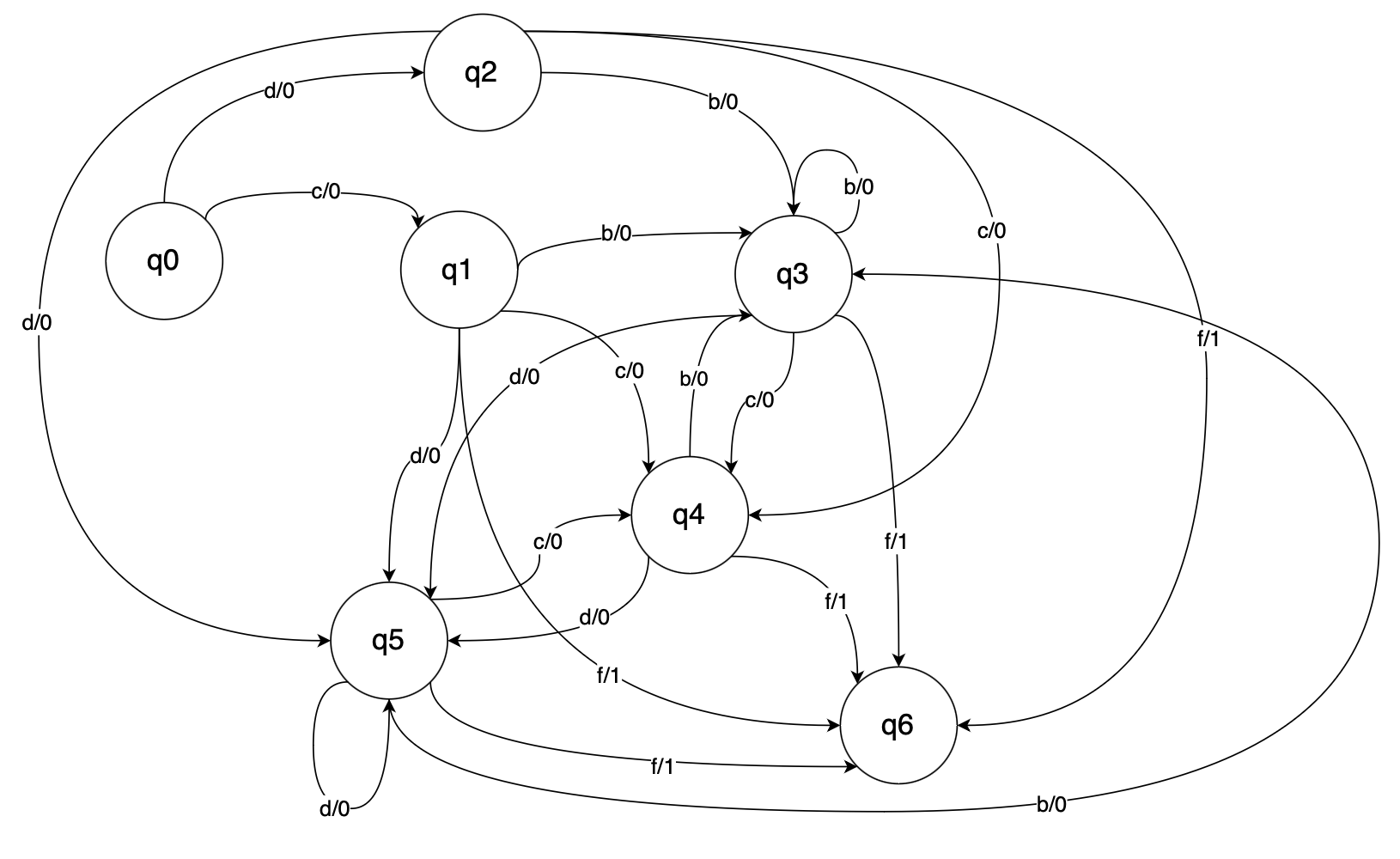


Рис. 1 - Диаграмма переходов

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

1. 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.

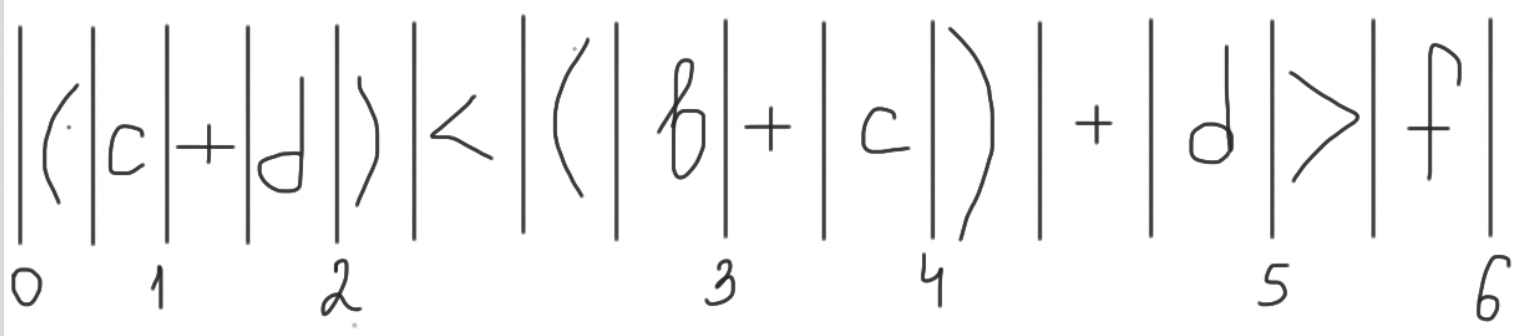


Рис. 1. — Разметка начального места и основных мест

1. Номер каждого основного места а присваивают всем неосновным местам, подчиненным месту а. 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.

Изображение выглядит как Шрифт, рукописный текст, линия, белый

Автоматически созданное описание

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

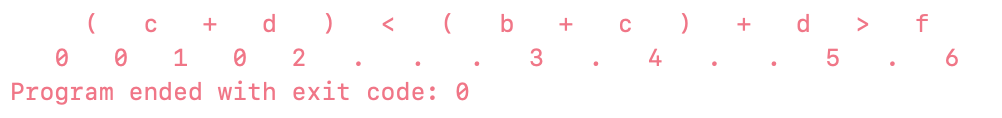


Рис. 2.2 — Программная реализация первого правила подчинения мест в регулярном выражении

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

Изображение выглядит как рукописный текст, текст, Шрифт, линия

Автоматически созданное описание

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

Изображение выглядит как Шрифт, текст, линия

Автоматически созданное описание

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

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

Изображение выглядит как текст, рукописный текст, Шрифт, линия

Автоматически созданное описание

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

Изображение выглядит как Шрифт, линия, текст

Автоматически созданное описание

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

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

Изображение выглядит как текст, рукописный текст, Шрифт, линия

Автоматически созданное описание

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

Изображение выглядит как линия, Шрифт

Автоматически созданное описание

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

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

1. 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 *sr* 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:

Изображение выглядит как текст, рукописный текст, Шрифт, линия

Автоматически созданное описание

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

1. Текстовые файлы входных и выходных данных программы:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рис. 8 — Текстовый файл входных данных

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рис. 9 — Текстовый файл выходных данных

1. Вывод

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

**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;

}