МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

ОТЧЁТ ИЗ ТРЁХ ЧАСТЕЙ ПО ТЕМАМ АЛГОРИТМЫ МАРКОВА И МАШИНА ТЬЮРИНГА

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил:
Студент группы ИВТ-21-2б
Безух Владимир Сергеевич

Проверил:
Доцент кафедры ИТАС
Полякова Ольга Андреевна

Пермь,
Октябрь 2021

Часть 1: алгоритмы Маркова. Постановка задачи

Создать эмулятор для работы с нормальными алгоритмами Маркова. Решить предложенные задачи.

Общий анализ задачи

Любое правило Маркова (ПМ) состоит из трёх частей: левой стороны правила, правой стороны правила и терминального статуса. Для удобства хранения информации о ПМ и дальнейшей обработки следует создать структуру данных MarkovRule, состоящую из трёх ранее названных компонентов. Значения левой и правой сторон ПМ сохраним в двух соответствующих объектах left_side_rule, right_side_rule класса string, а терминальный статус в переменной is_rule_terminal типа bool (если true, — значит правило терминальное).

Для хранения набора ПМ достаточно использовать объект vector<MarkovRule> класса vector, состоящий из значений структуры данных MarkovRule.

Для поиска позиции первого вхождения position под-слова из левой стороны ПМ в обрабатываемом слове word достаточно использовать метод word.find(left_side_rule). Если такое вхождение найдено (т.е. найденная методом .find() позиция отлична от string::npos), для применения ПМ достаточно использовать метод word.replace(position, left_side_rule.length(), right_side_rule).

С помощью двух вложенных циклов можно обработать весь набор ПМ целиком. Внутренний цикл отвечает за последовательную проверку и применение ПМ на конкретном шаге нормального алгоритма Маркова (НАМ), внешний цикл отвечает за пошаговую работу НАМ в целом. При применении терминального ПМ или в отсутствии применения каждого ПМ из набора на конкретном шаге, следует немедленно прекратить выполнение НАМ.

Остальной исходный код программы представляет собой несколько взаимосвязанных функций. Все участки кода декомпозированы по принципам ООП. Отдельные функции для обработки предложенных задач отвечают за инициализацию конкретных наборов ПМ. Эти функции не являются основной частью эмулятора машины Маркова.

Данная программа для выполнения НАМ универсальна, дискретна, конечна, результативна, выполнима, детерминирована и последовательна. С учётом особенностей ввода и хранения обрабатываемого слова объекта word класса string данную программу можно применять для обработки любых слов любым конечным набором ПМ.

В программе отсутствует модуль проверки правильности введённой пользователем строки символов и модуль обработки ошибок выполнения, поэтому предполагается, что введённое пользователем на обработку слово помещается в объект word класса string и состоит только из символов задействованного алфавита НАМ. При вводе строки, нарушающей эти условия, корректность выполнения программы не гарантируется.

Демонстрация исполнения кода на примере одной из задач

В рамках демонстрации работы кода его участки расположены в удобной для восприятия последовательности. Расположение этих участков в исходном коде может отличаться в соответствии со всеми особенностями языка программирования C++.

```
#include <vector>
#include <string>
#include <iomanip>
#include <iostream>

using namespace std;

int main()
{
	setlocale(LC_ALL, "Russian");

	firstNormalMarkovAlgorithmTask();
}
```

Выше приведённый участок кода содержит директивы подключения необходимых в работе программы стандартных библиотек, объявление рабочего пространства имён, установку русской локализации обработки символов, функцию обработки первой из предложенных для решения задач на тему НАМ.

```
struct MarkovRule
{
	bool is_rule_terminal;
	string left_side_rule, right_side_rule;

	MarkovRule(string left_side_rule, bool is_rule_terminal, string right_side_rule) :
			left_side_rule(left_side_rule), is_rule_terminal(is_rule_terminal),
			right_side_rule(right_side_rule) {}
};
```

Для понимания устройства кода следует рассмотреть структуру MarkovRule. Структура содержит в себе два значения для левой и правой сторон ПМ, хранящихся в соответствующих объектах left_side_rule, right_side_rule класса string; а также терминальный статус в переменной is_rule_terminal типа bool (если true, — значит правило терминальное). Конструктор выполнен таким образом, чтобы обеспечить максимально возможную абстракцию по отношению к стандартной записи ПМ (левая часть ПМ, знак терминального статуса, правая часть ПМ).

В комментарии к функции указана постановка и решение предлагаемой задачи.

```
void firstNormalMarkovAlgorithmTask()
{
	/*
	* Дано:
	* A = {a, b}.
	* Удалить из непустого произвольного слова P его первый символ.
	* Пустое слово не менять.
	*
	* Решение:
	* *a |→
	* *b |→
	*  * |→
	*    → *
	*/
```

```
const vector<MarkovRule> MARKOV_RULES = { MarkovRule("*a", true, ""),
                                          MarkovRule("*b", true, ""),
                                          MarkovRule("*", true, ""),
                                          MarkovRule("", false, "*") };

        cout << "Введите строку для первого задания (A = {a, b}): ";
        normalMarkovAlgorithmTaskInterface(MARKOV_RULES);
}
```

Затем происходит передача по ссылке константного объекта класса vector MARKOV_RULES, содержащего объекты структуры MarkovRule. Объекты структуры MarkovRule объявляются и инициализируются через простой конструктор в соответствии с набором ПМ для решения поставленной задачи.

```
void normalMarkovAlgorithmTaskInterface(const vector<MarkovRule> &MARKOV_RULES)
{
        /*
        * Запрашиваем строку INPUT_WORD у пользователя, обрабатываем введённую строку
        * INPUT_WORD по набору правил нормального алгоритма Маркова MARKOV_RULES,
        * выводим результирующую строку OUTPUT_WORD.
        */

        string INPUT_WORD; getline(cin, INPUT_WORD); cout << endl;

        const string OUTPUT_WORD =
                normalMarkovAlgorithm(INPUT_WORD, MARKOV_RULES);

        cout << endl << "Результирующая строка: " << OUTPUT_WORD << "\n\n";
}
```

Комментарий в коде функции normalMarkovAlgorithmTaskInterface() ёмко и достаточно описывает происходящее в ней.

```
const string normalMarkovAlgorithm(const string &WORD,
                                    const vector<MarkovRule> &MARKOV_RULES)
{
        string output_word = WORD;
        size_t step_counter = 0; // подсчёт количества шагов алгоритма
        bool is_terminal_flag = false; // флаг для терминального правила

        ...
```

Функция normalMarkovAlgorithm() возвращает результирующую строку. Дальнейшей обработки результата не предполагается, поэтому он хранится в константном объекте класса string. В качестве параметров функция принимает по ссылке объект класса string в виде обрабатываемого слова и объект класса vector в виде набора ПМ.

```
        ...

        // до тех пор, пока не выполнилось терминальное правило...
        while (!is_terminal_flag)
        {
                size_t number_of_rule = 1; // счётчик для номера правила
                bool has_replacement_occured = false; // флаг об успешном применении правила

                // последовательно проходим по всем правилам
                for (const auto &MARKOV_RULE : MARKOV_RULES)
                {
                        // если ранее не применялось терминальное правило, то...
                        if (!is_terminal_flag)
                        {
                                ++step_counter;
```

```
                        // попробовать применить правило и сообщить результат
                        has_replacement_occured =
                                applyMarkovRule(output_word, MARKOV_RULE);

                        /* вывод подробной информации о состоянии слова после попытки
                        применить правило */
                        cout << "Строка после применения правила №"
                        << number_of_rule << ":\t"
                        << setiosflags(ios::left) << setw(20)
                        << output_word << resetiosflags(ios::left)
                        << "\t(" << step_counter << " шаг)\n";

                        // если удалось применить правило, то...
                if (has_replacement_occured)
                        {
                        // передать флагу терминальный статус ранее применённого правила
                                is_terminal_flag = MARKOV_RULE.is_rule_terminal;
                                break;
                        }
                }
                else { break; } // если уже применили правило, выходим из цикла

                ++number_of_rule; // переходим к следующему правилу
        }

        // если ни одно правило не было применено, завершаем алгоритм
        if (!has_replacement_occured)
                is_terminal_flag = true;
}
```

Дальнейшие действия в коде выше раскрываются в соответствующих комментариях и не требуют дополнительных объяснений. В качестве альтернативной реализации, вместо флага для обработки состояния выполнения программы можно использовать операторы break и continue.

```
bool applyMarkovRule(string &word, const MarkovRule &MARKOV_RULE)
{
        /* поиск в обрабатываемом слове
        позиции первого вхождения под-слова из левой части правила */
        unsigned int position = word.find(MARKOV_RULE.left_side_rule);

        if (position != string::npos) // если было найдено вхождение, то...
        {
                // заменяем найденное вхождение на правую часть правила, возвращаем успех
                word.replace(position, MARKOV_RULE.left_side_rule.length(),
                                MARKOV_RULE.right_side_rule);
                return true;
        }

        return false; // если не применили правило, возвращаем неуспех
}
```

Функция applyMarkovRule() возвращает логическое значение bool в зависимости от результата исполнения: true — если правило было применено, false — если этого не произошло. Первый параметр функции не константный, потому что идейно функция обрабатывает переданное значение сразу напрямую. Других пояснений по коду не требуется.

Описание переменных

struct MarkovRule
{
    bool is_rule_terminal;
    string left_side_rule, right_side_rule;

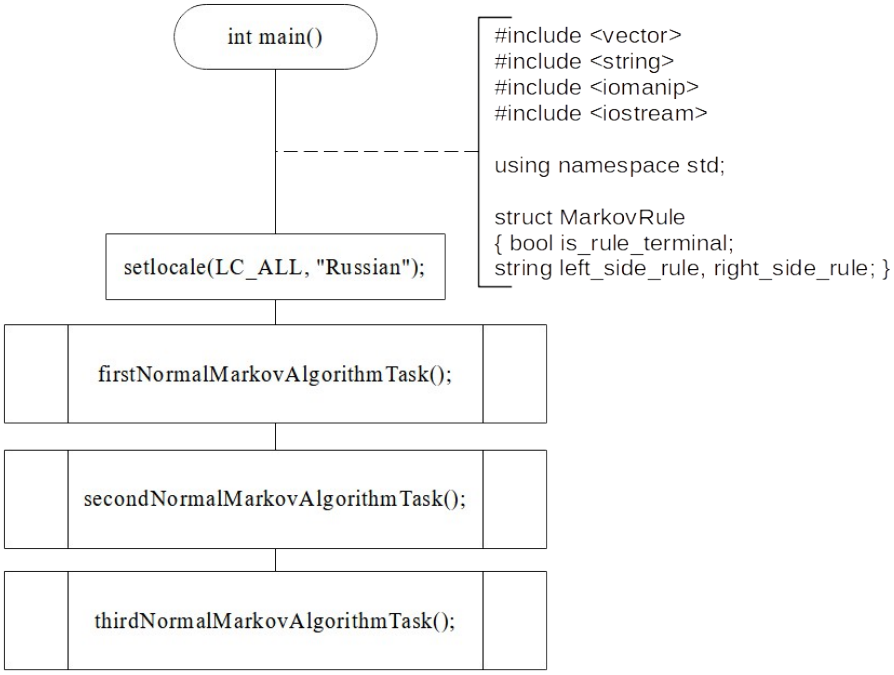
    MarkovRule(string left_side_rule, bool is_rule_terminal, string right_side_rule) :
    left_side_rule(left_side_rule),
    is_rule_terminal(is_rule_terminal),
    right_side_rule(right_side_rule) {}
}; — строение структуры: три переменные и конструктор.

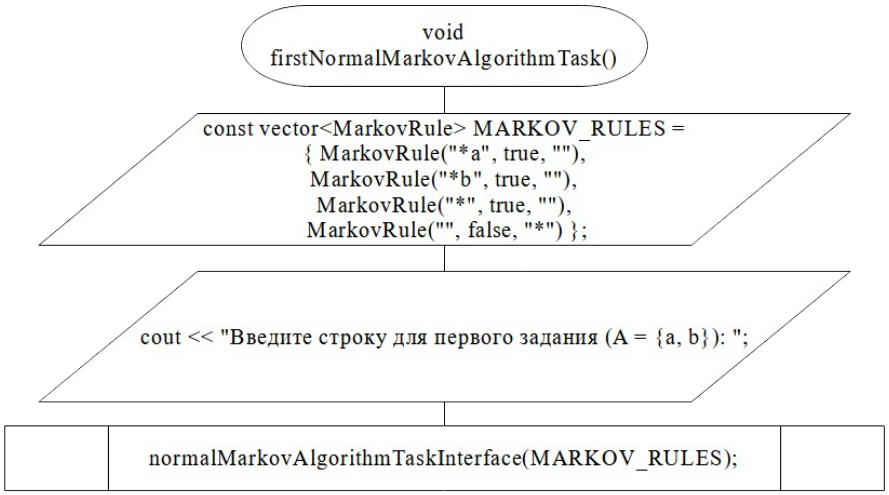
const vector<MarkovRule> MARKOV_RULES = { MarkovRule("", true, "") }; — константный вектор для набора ПМ, проинициализированный конструктором(ами) объектов структуры MarkovRule.

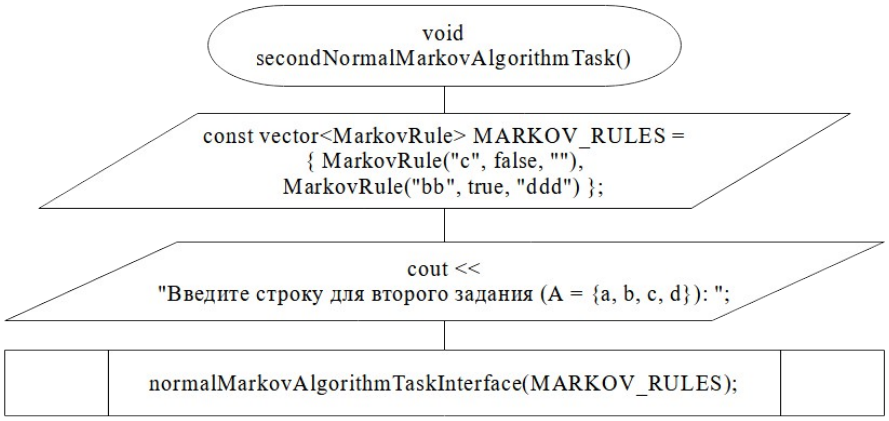
string INPUT_WORD; getline(cin, INPUT_WORD); — введённое для обработки слово.

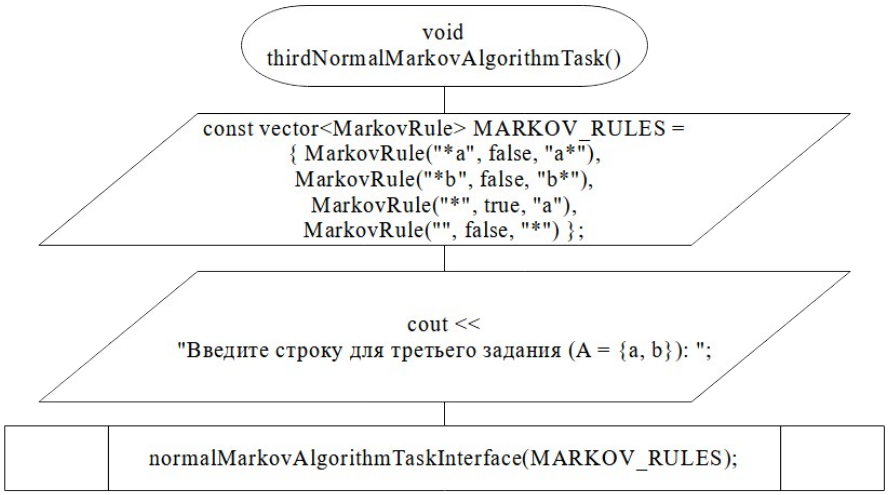
const string OUTPUT_WORD =
normalMarkovAlgorithm(INPUT_WORD, MARKOV_RULES); — результирующее слово.

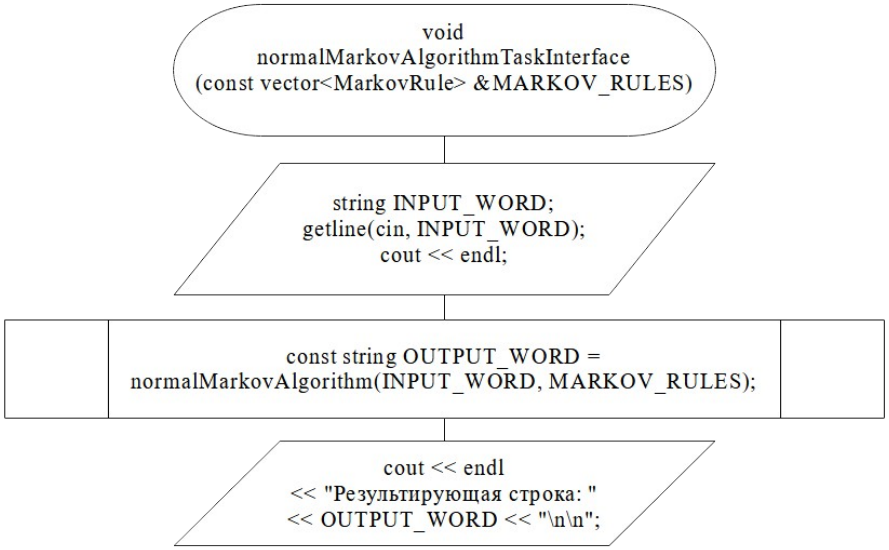
Блок-схема

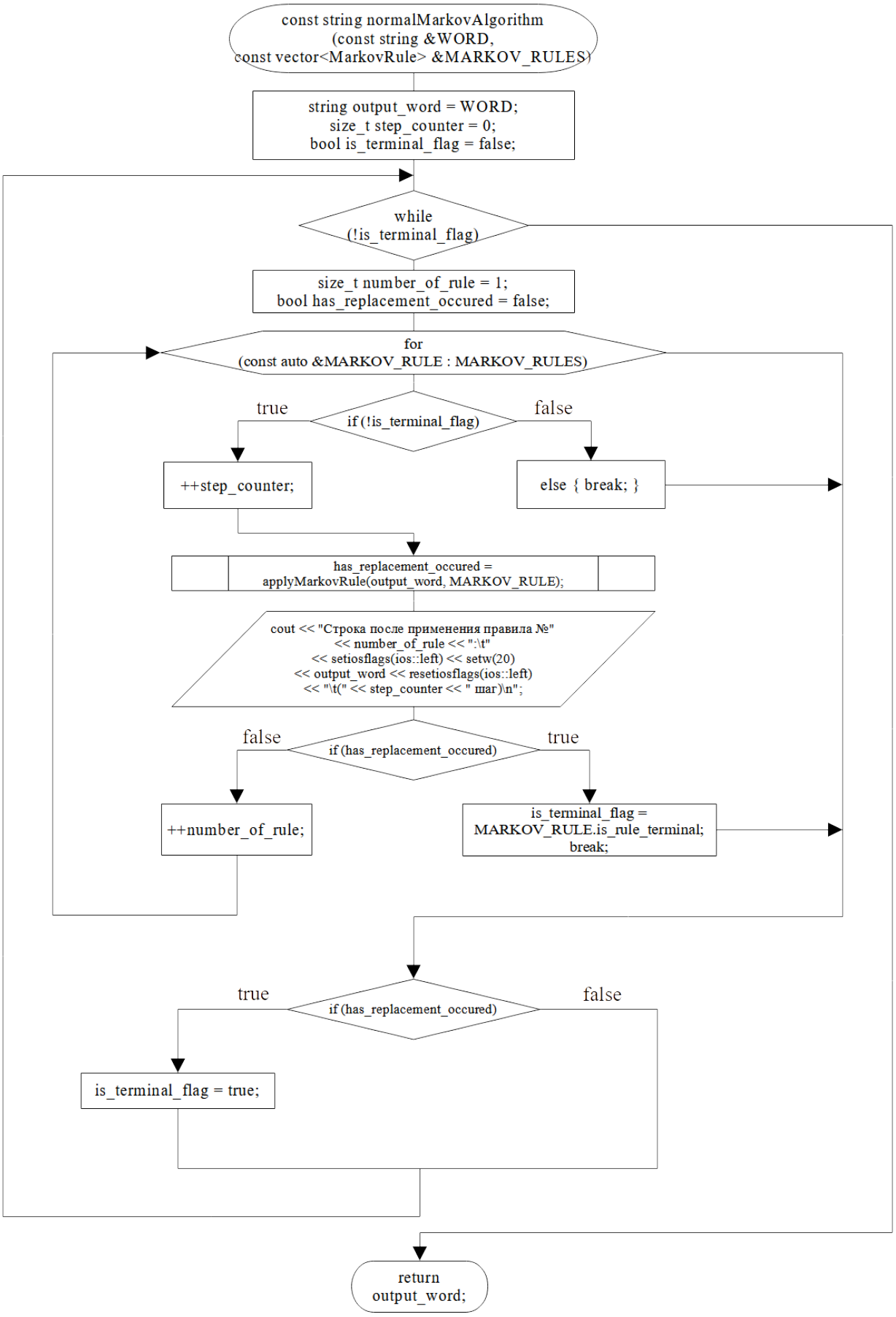

int main()
#include <vector>
#include <string>
#include <iomanip>
#include <iostream>
using namespace std;
struct MarkovRule
{ bool is_rule_terminal;
string left_side_rule, right_side_rule; }
setlocale(LC_ALL, "Russian");
firstNormalMarkovAlgorithmTask();
secondNormalMarkovAlgorithmTask();
thirdNormalMarkovAlgorithmTask();

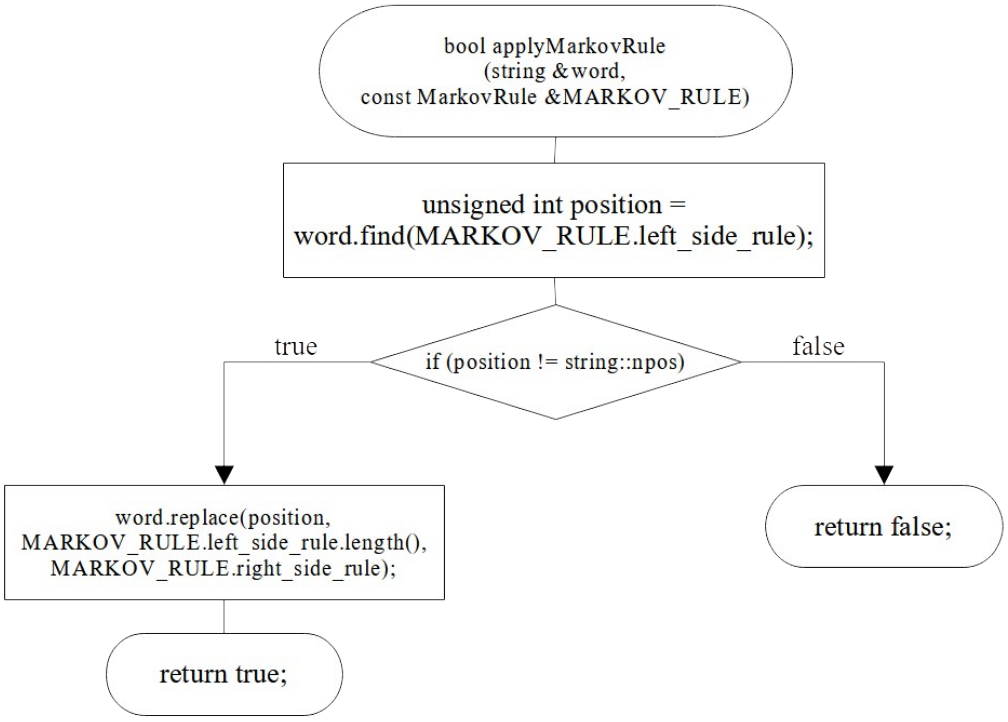

void
firstNormalMarkovAlgorithmTask()
const vector<MarkovRule> MARKOV_RULES =
{ MarkovRule("*a", true, ""),
MarkovRule("*b", true, ""),
MarkovRule("*", true, ""),
MarkovRule("", false, "*") };
cout << "Введите строку для первого задания (A = {a, b}): ";
normalMarkovAlgorithmTaskInterface(MARKOV_RULES);


void
secondNormalMarkovAlgorithmTask()
const vector<MarkovRule> MARKOV_RULES =
{ MarkovRule("c", false, ""),
MarkovRule("bb", true, "ddd") };
cout <<
"Введите строку для второго задания (A = {a, b, c, d}): ";
normalMarkovAlgorithmTaskInterface(MARKOV_RULES);


void
thirdNormalMarkovAlgorithmTask()
const vector<MarkovRule> MARKOV_RULES =
{ MarkovRule("*a", false, "a*"),
MarkovRule("*b", false, "b*"),
MarkovRule("*", true, "a"),
MarkovRule("", false, "*") };
cout <<
"Введите строку для третьего задания (A = {a, b}): ";
normalMarkovAlgorithmTaskInterface(MARKOV_RULES);


void
normalMarkovAlgorithmTaskInterface
(const vector<MarkovRule> &MARKOV_RULES)
string INPUT_WORD;
getline(cin, INPUT_WORD);
cout << endl;
const string OUTPUT_WORD =
normalMarkovAlgorithm(INPUT_WORD, MARKOV_RULES);
cout << endl
<< "Результирующая строка: "
<< OUTPUT_WORD << "\n\n";


const string normalMarkovAlgorithm
(const string &WORD,
const vector<MarkovRule> &MARKOV_RULES)
string output_word = WORD;
size_t step_counter = 0;
bool is_terminal_flag = false;
while
(!is_terminal_flag)
size_t number_of_rule = 1;
bool has_replacement_occured = false;
for
(const auto &MARKOV_RULE : MARKOV_RULES)
true
if (!is_terminal_flag)
false
++step_counter;
else { break; }
has_replacement_occured =
applyMarkovRule(output_word, MARKOV_RULE);
cout << "Строка после применения правила №"
<< number_of_rule << ":\t"
<< setiosflags(ios::left) << setw(20)
<< output_word << resetiosflags(ios::left)
<< "\t(" << step_counter << " шаг)\n";
false
if (has_replacement_occured)
true
++number_of_rule;
is_terminal_flag =
MARKOV_RULE.is_rule_terminal;
break;
true
if (has_replacement_occured)
false
is_terminal_flag = true;
return
output_word;


bool applyMarkovRule
(string &word,
const MarkovRule &MARKOV_RULE)
unsigned int position =
word.find(MARKOV_RULE.left_side_rule);
if (position != string::npos)
true
false
word.replace(position,
MARKOV_RULE.left_side_rule.length(),
MARKOV_RULE.right_side_rule);
return false;
return true;

Исходный код

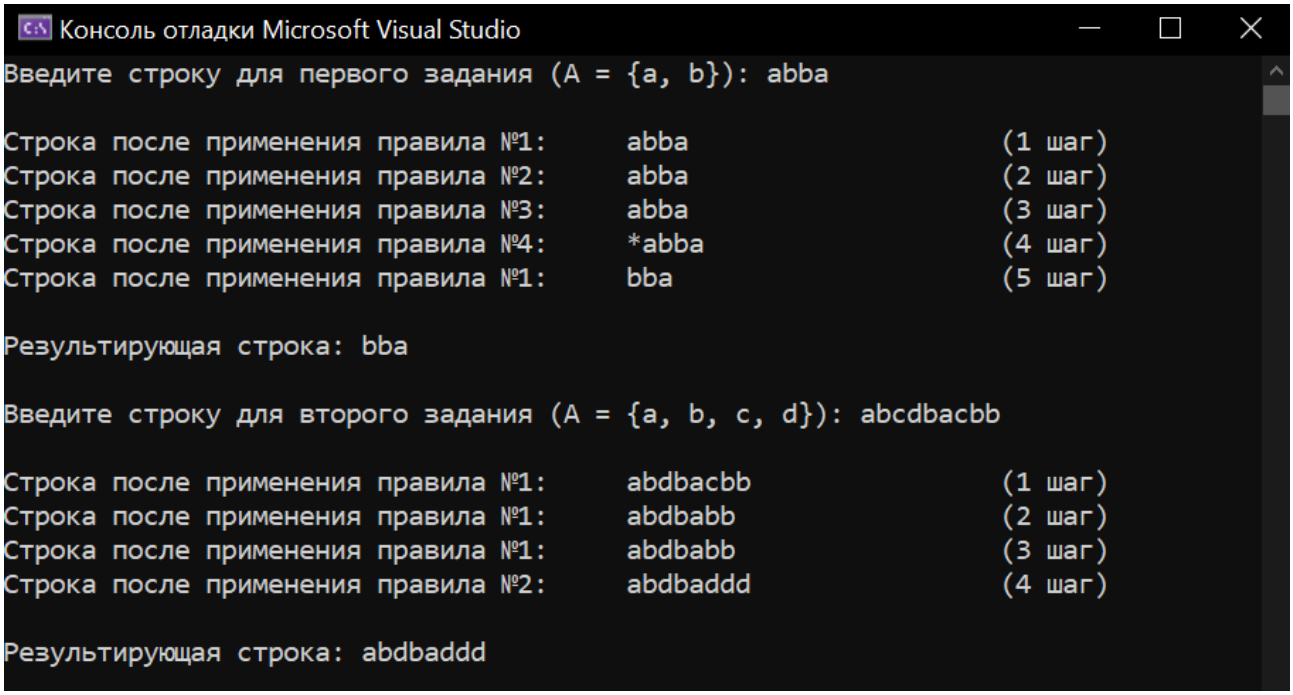
```
/*
* This code is licensed under the Creative Commons
* Attribution - NonCommercial - NoDerivatives 4.0 International License.
* To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/
* or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
*
* https://bezukh.wixsite.com/blog
* https://github.com/BezukhVladimir
*
* © Developed by Bezukh Vladimir. All right reserved.
*/

/*
* Developed by Bezukh Vladimir
* October 2021
* ИВТ-21-26
*
* Консольное приложение для применения нормального алгоритма Маркова.
*/

#include <vector>
#include <string>
#include <iomanip>
#include <iostream>

using namespace std;

struct MarkovRule
{
	bool is_rule_terminal;
	string left_side_rule, right_side_rule;

	MarkovRule(string left_side_rule, bool is_rule_terminal, string right_side_rule) :
			left_side_rule(left_side_rule), is_rule_terminal(is_rule_terminal),
			right_side_rule(right_side_rule) {}
};

bool applyMarkovRule(string &word, const MarkovRule &MARKOV_RULE)
{
	/* поиск в обрабатываемом слове
	позиции первого вхождения под-слова из левой части правила */
	unsigned int position = word.find(MARKOV_RULE.left_side_rule);

	if (position != string::npos) // если было найдено вхождение, то...
	{
		// заменяем найденное вхождение на правую часть правила, возвращаем успех
		word.replace(position, MARKOV_RULE.left_side_rule.length(),
				MARKOV_RULE.right_side_rule);
		return true;
	}

	return false; // если не применили правило, возвращаем неуспех
}
```

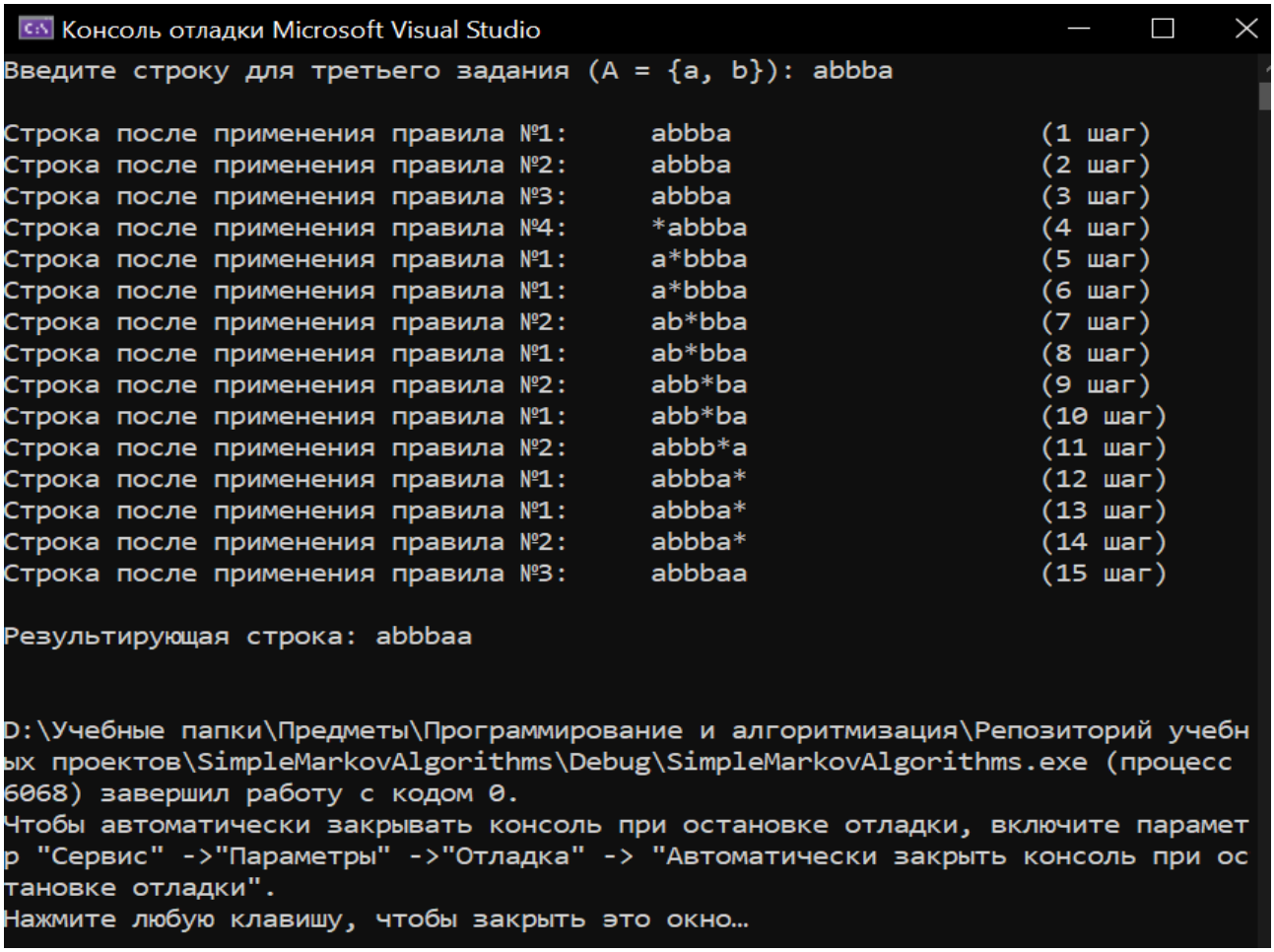
```
const string normalMarkovAlgorithm(const string &WORD,
                                   const vector<MarkovRule> &MARKOV_RULES)
{
	string output_word = WORD;
	size_t step_counter = 0; // подсчёт количества шагов алгоритма
	bool is_terminal_flag = false; // флаг для терминального правила


	// до тех пор, пока не выполнилось терминальное правило...
	while (!is_terminal_flag)
	{
		size_t number_of_rule = 1; // счётчик для номера правила
		bool has_replacement_occured = false; // флаг об успешном применении правила

		// последовательно проходим по всем правилам
		for (const auto &MARKOV_RULE : MARKOV_RULES)
		{
			// если ранее не применялось терминальное правило, то...
			if (!is_terminal_flag)
			{
				++step_counter;

				// попробовать применить правило и сообщить результат
				has_replacement_occured =
					applyMarkovRule(output_word, MARKOV_RULE);

				/* вывод подробной информации о состоянии слова после попытки
				применить правило */
				cout << "Строка после применения правила №"
				<< number_of_rule << ":\t"
				<< setiosflags(ios::left) << setw(20)
				<< output_word << resetiosflags(ios::left)
				<< "\t(" << step_counter << " шаг)\n";

				// если удалось применить правило, то...
			if (has_replacement_occured)
				{
				// передать флагу терминальный статус ранее применённого правила
					is_terminal_flag = MARKOV_RULE.is_rule_terminal;
					break;
				}
			}
			else { break; } // если уже применили правило, выходим из цикла

			++number_of_rule; // переходим к следующему правилу
		}

		// если ни одно правило не было применено, завершаем алгоритм
		if (!has_replacement_occured)
			is_terminal_flag = true;
	}

	return output_word;
}
```

```
void normalMarkovAlgorithmTaskInterface(const vector<MarkovRule> &MARKOV_RULES)
{
	/*
	* Запрашиваем строку INPUT_WORD у пользователя, обрабатываем введённую строку
	* INPUT_WORD по набору правил нормального алгоритма Маркова MARKOV_RULES,
	* выводим результирующую строку OUTPUT_WORD.
	*/

	string INPUT_WORD; getline(cin, INPUT_WORD); cout << endl;

	const string OUTPUT_WORD =
		normalMarkovAlgorithm(INPUT_WORD, MARKOV_RULES);

	cout << endl << "Результирующая строка: " << OUTPUT_WORD << "\n\n";
}

void firstNormalMarkovAlgorithmTask()
{
	/*
	* Дано:
	* A = {a, b}.
	* Удалить из непустого произвольного слова P его первый символ.
	* Пустое слово не менять.
	*
	* Решение:
	* *a |→
	* *b |→
	*  * |→
	*    → *
	*/

	const vector<MarkovRule> MARKOV_RULES = { MarkovRule("*a", true, ""),
	                                          MarkovRule("*b", true, ""),
	                                          MarkovRule("*", true, ""),
	                                          MarkovRule("", false, "*") };

	cout << "Введите строку для первого задания (A = {a, b}): ";
	normalMarkovAlgorithmTaskInterface(MARKOV_RULES);
}

void secondNormalMarkovAlgorithmTask()
{
	/*
	* Дано:
	* A = {a, b, c, d}.
	* В произвольном слове P требуется удалить все вхождения символа c,
	* затем заменить первое вхождение подслова bb на ddd.
	*
	* Решение:
	*  c  →
	* bb |→ ddd
	*/

	const vector<MarkovRule> MARKOV_RULES = { MarkovRule("c", false, ""),
	                                          MarkovRule("bb", true, "ddd") };

	cout << "Введите строку для второго задания (A = {a, b, c, d}): ";
	normalMarkovAlgorithmTaskInterface(MARKOV_RULES);
}




void thirdNormalMarkovAlgorithmTask()
```

```
{
	/*
	* Дано: A = {a, b}.
	* Требуется приписать символ a к концу произвольного слова P.
	*
	* Решение:
	* *a  → a*
	* *b  → b*
	*  * |→ a
	*     → *
	*/

	const vector<MarkovRule> MARKOV_RULES = { MarkovRule("*a", false, "a*"),
	                                          MarkovRule("*b", false, "b*"),
	                                          MarkovRule("*", true, "a"),
	                                          MarkovRule("", false, "*") };

	cout << "Введите строку для третьего задания (A = {a, b}): ";
	normalMarkovAlgorithmTaskInterface(MARKOV_RULES);
}

int main()
{
	setlocale(LC_ALL, "Russian");

	firstNormalMarkovAlgorithmTask();
	secondNormalMarkovAlgorithmTask();
	thirdNormalMarkovAlgorithmTask();
}
```

Скриншоты консольного интерфейса программы





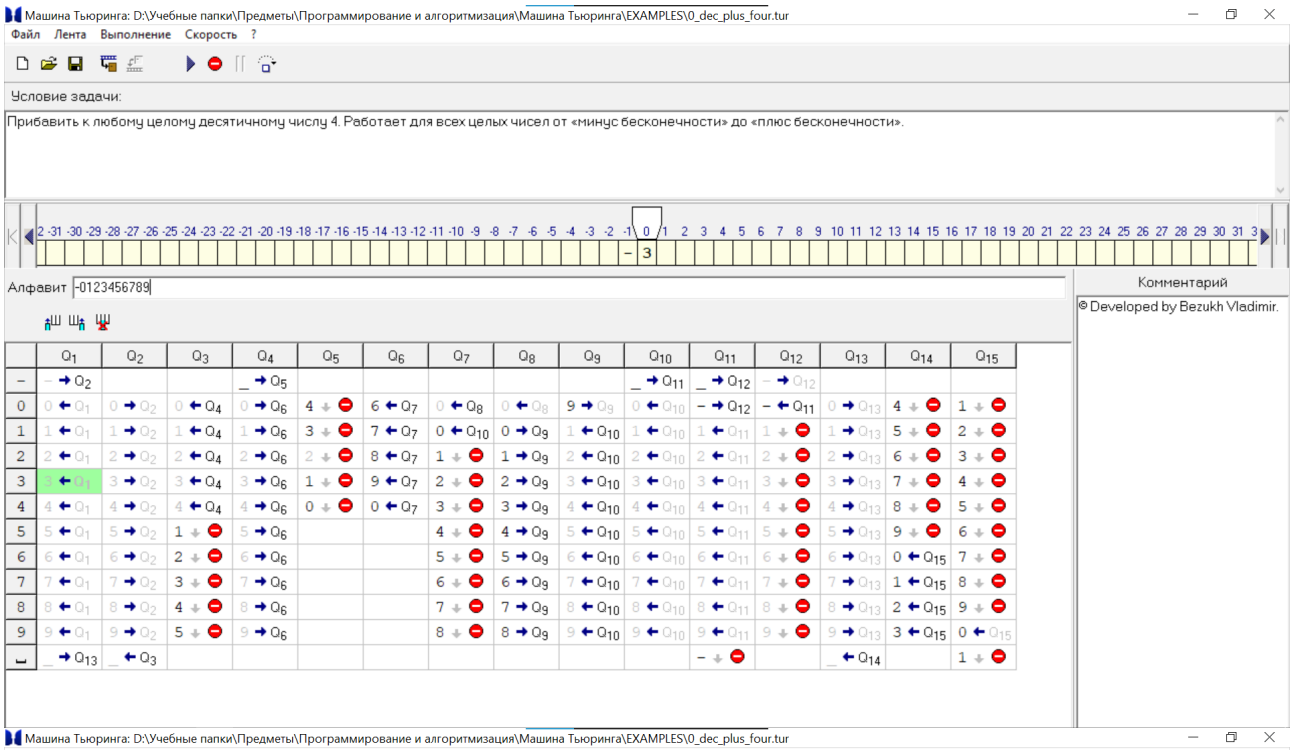
Анализ результатов

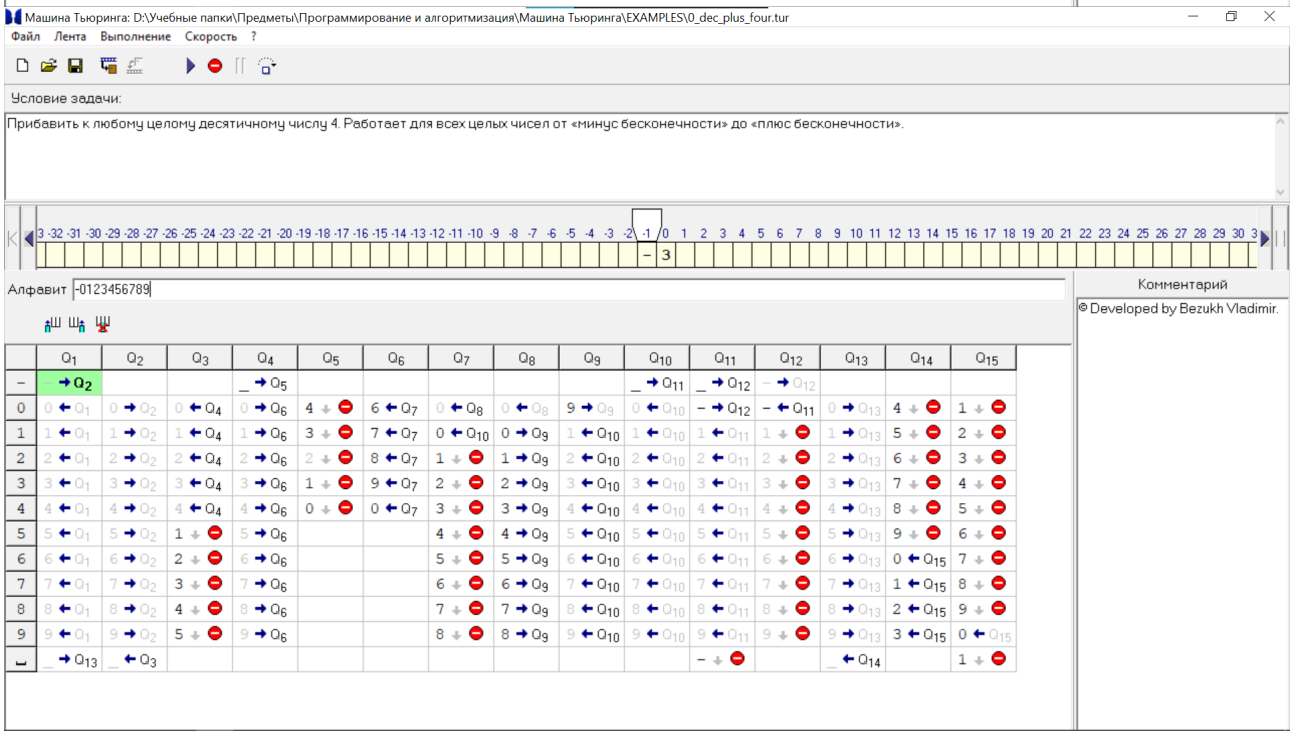
Эмулятор НАМ корректно отработал в соответствии с реализованным консольным интерфейсом программы и поддерживаемыми входными данными (ранее описано, что отсутствует модуль проверки вводимой пользователем последовательности символов). Результаты детерминированы и не требуют дополнительных пояснений.

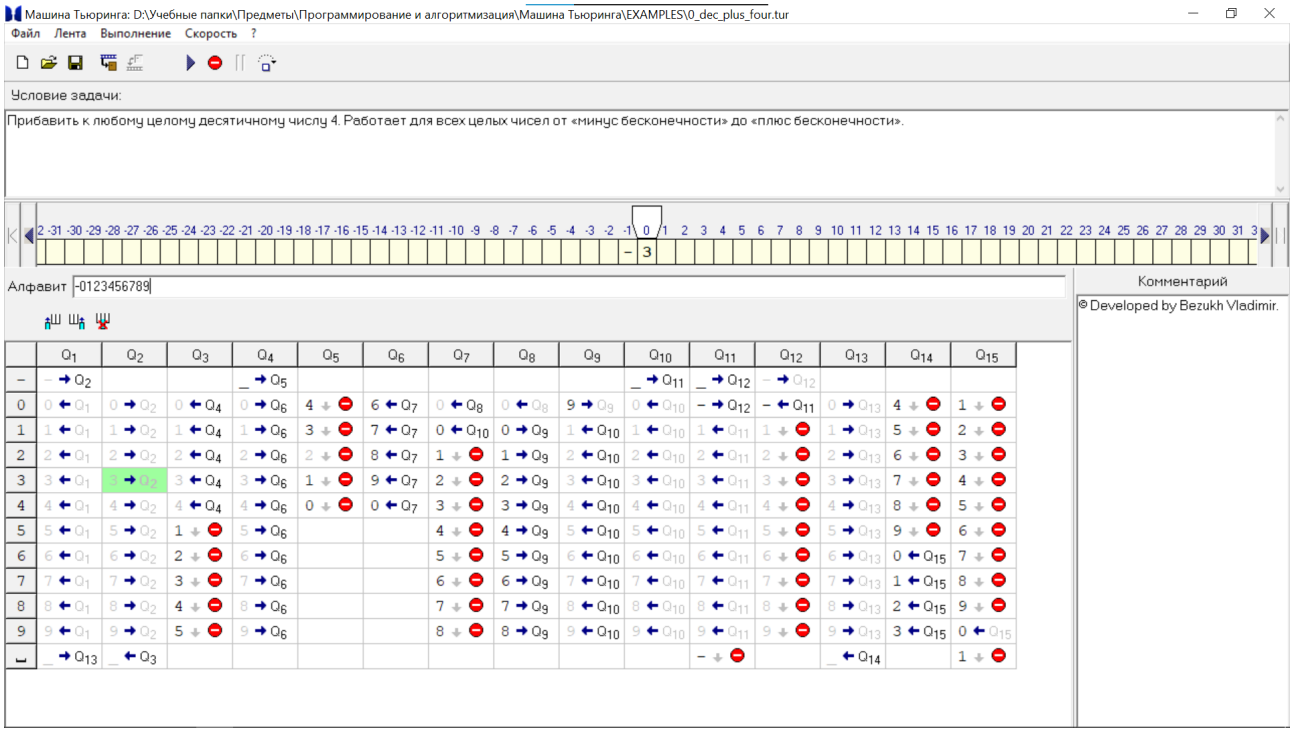
Часть 2: машина Тьюринга

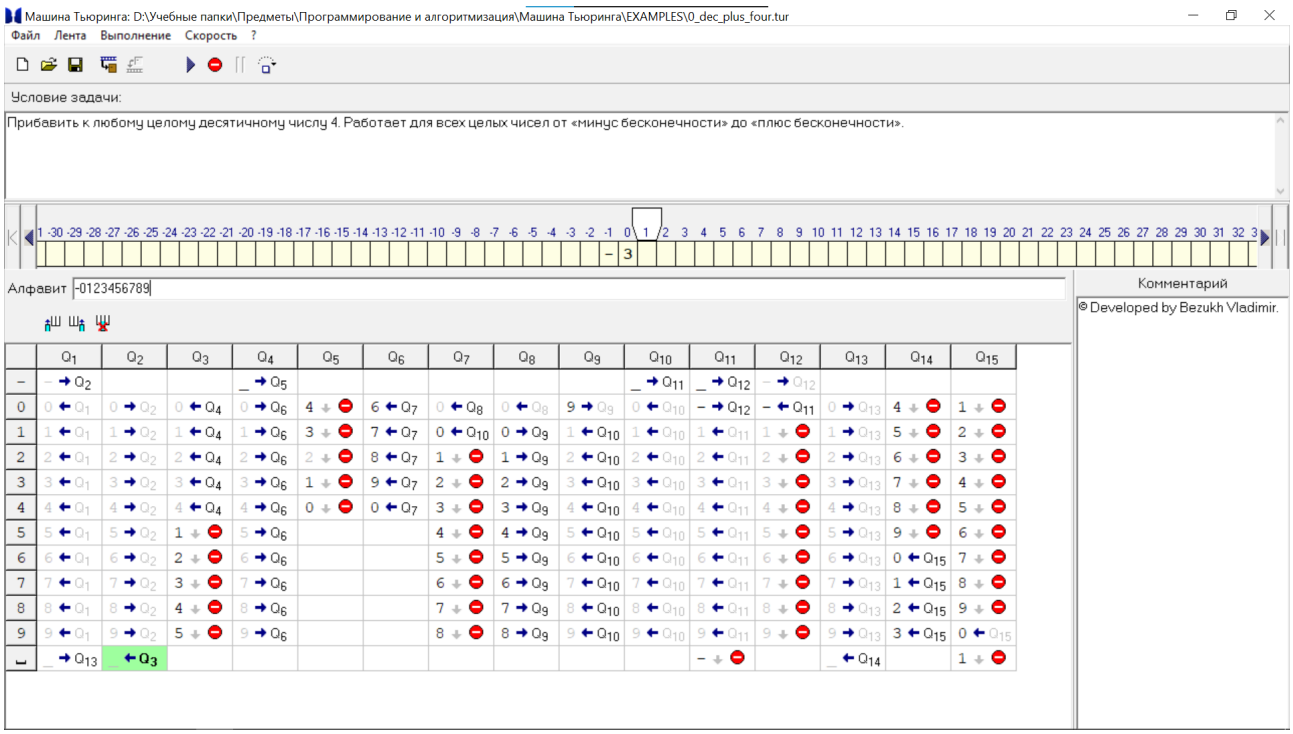
Первое задание:

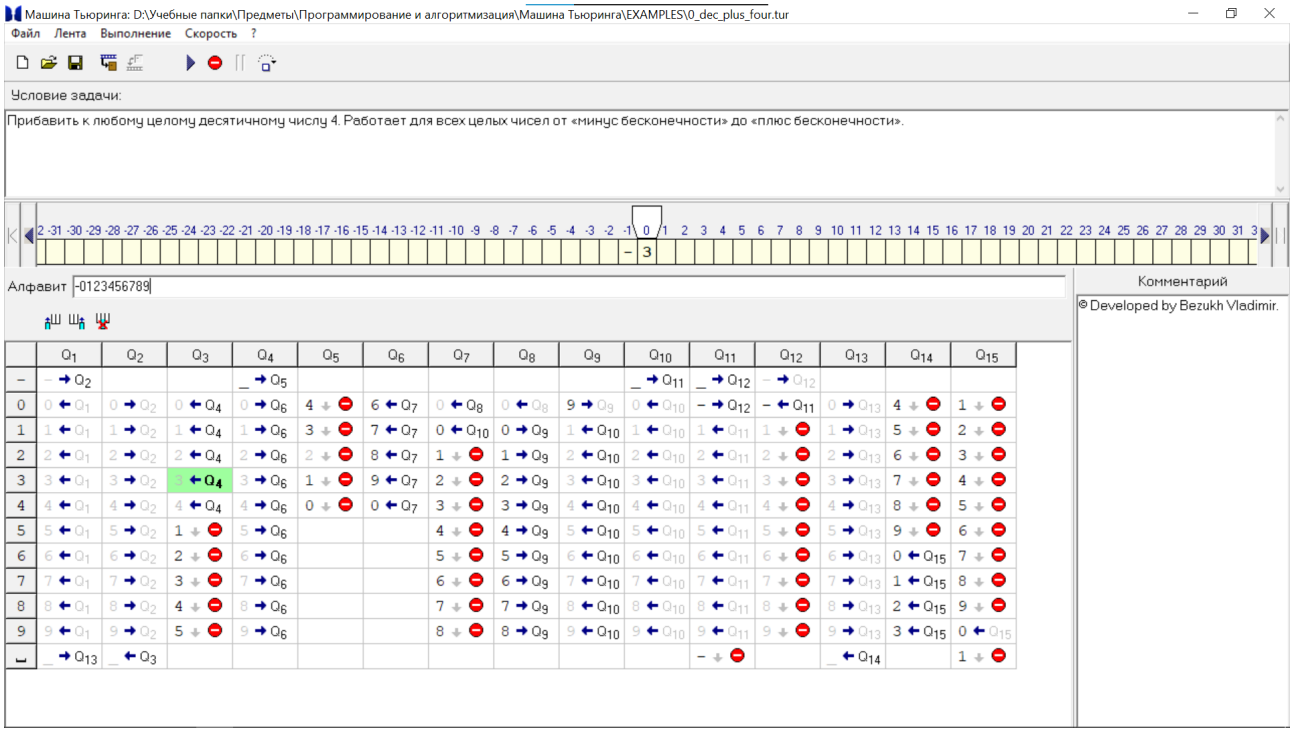
Постановка задачи указана на скриншоте. Сначала путём проверки крайнего левого символа определяется, является ли число отрицательным или положительным, затем, в зависимости от того, какое это число, применяются различные состояния и соответствующие им наборы правил обработки ячейки под головой машины. Учтены «сложные» случаи перехода исходного числа от отрицательного к положительному, все особенности переполнения разряда и переноса. Полная конфигурация машины для эмулятора Константина Полякова в файле формата .tur доступна в репозитории данного учебного проекта.

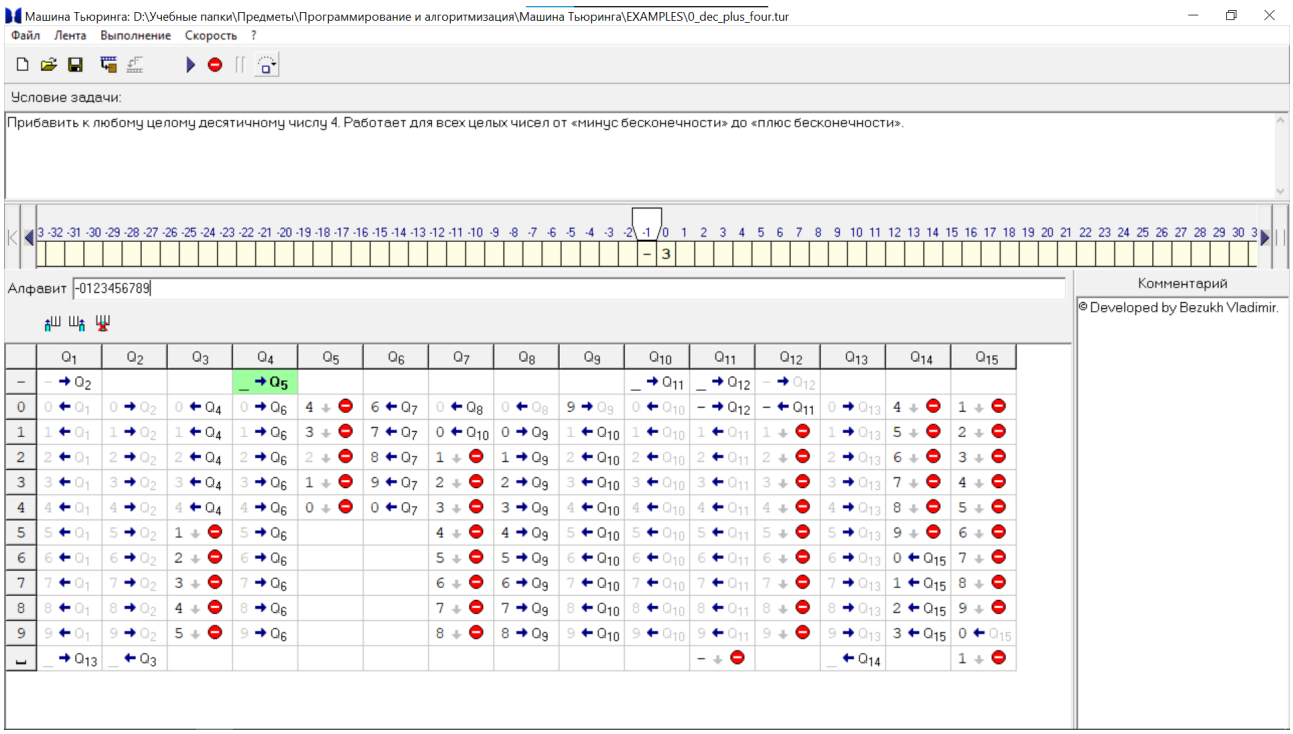


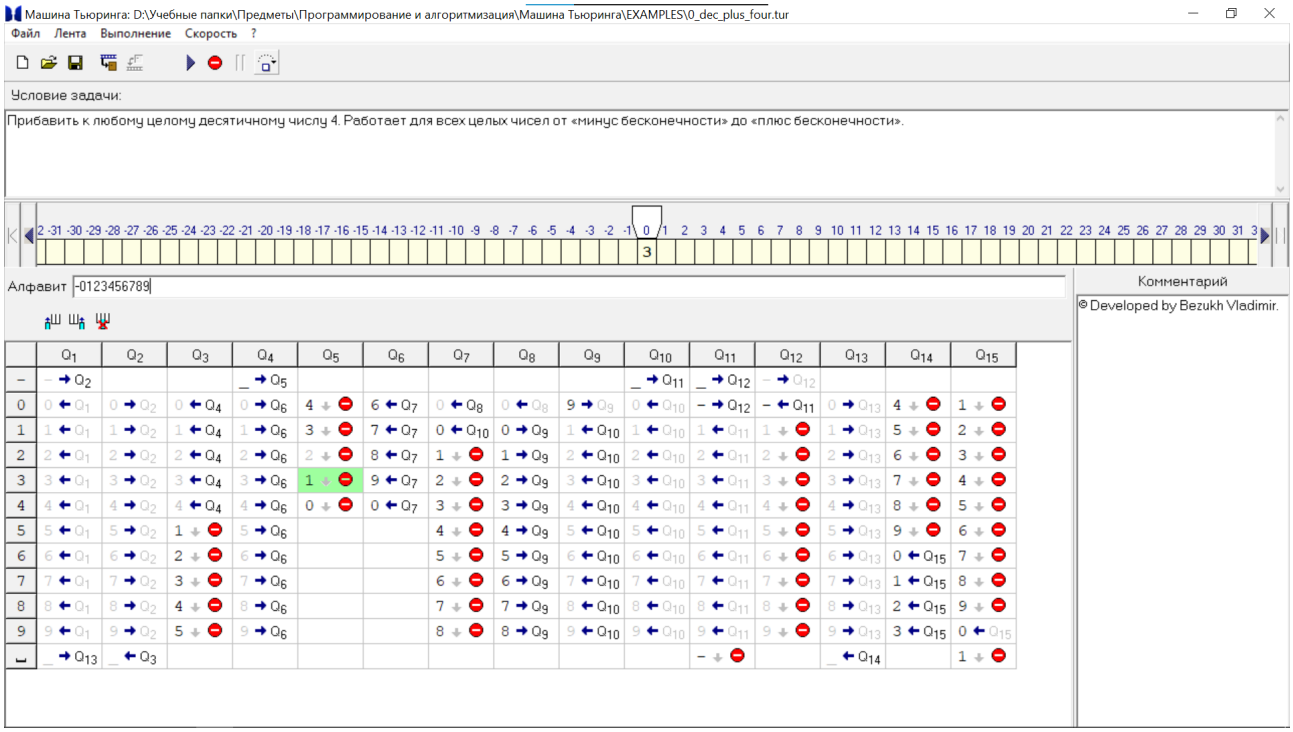


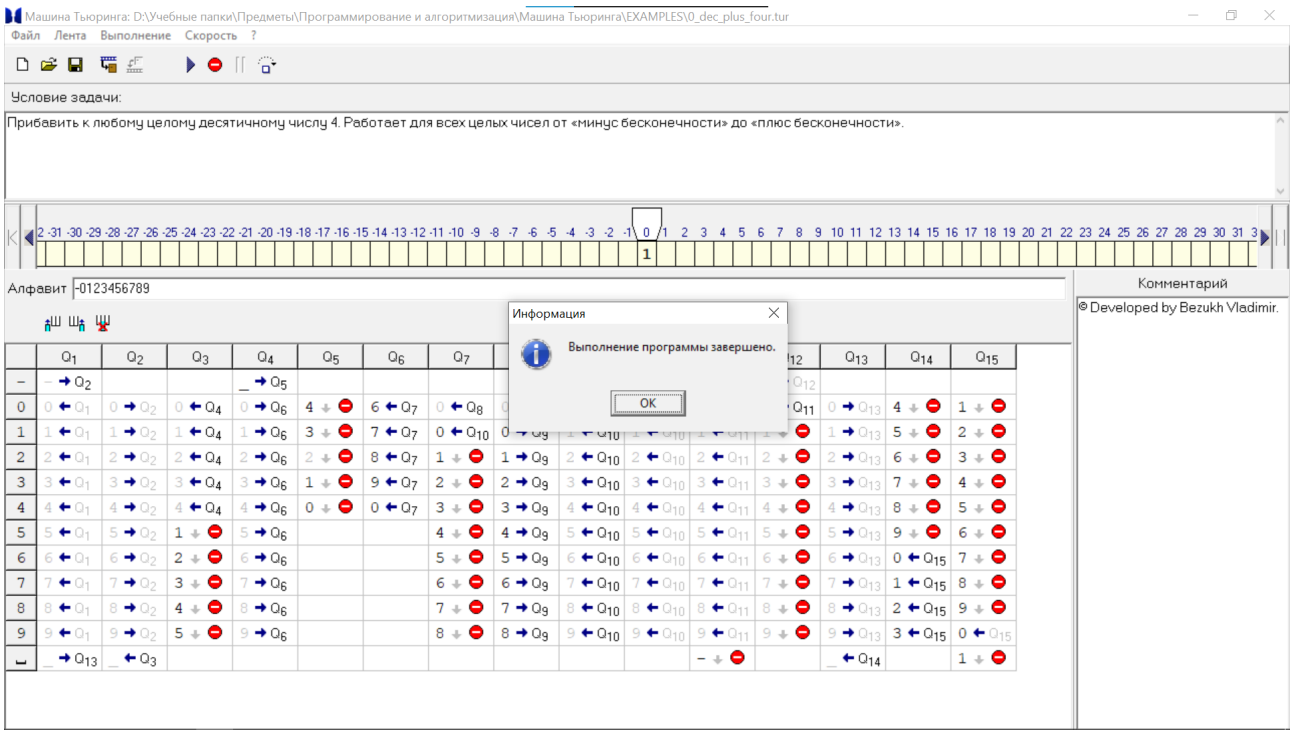

Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.


Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.


Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.

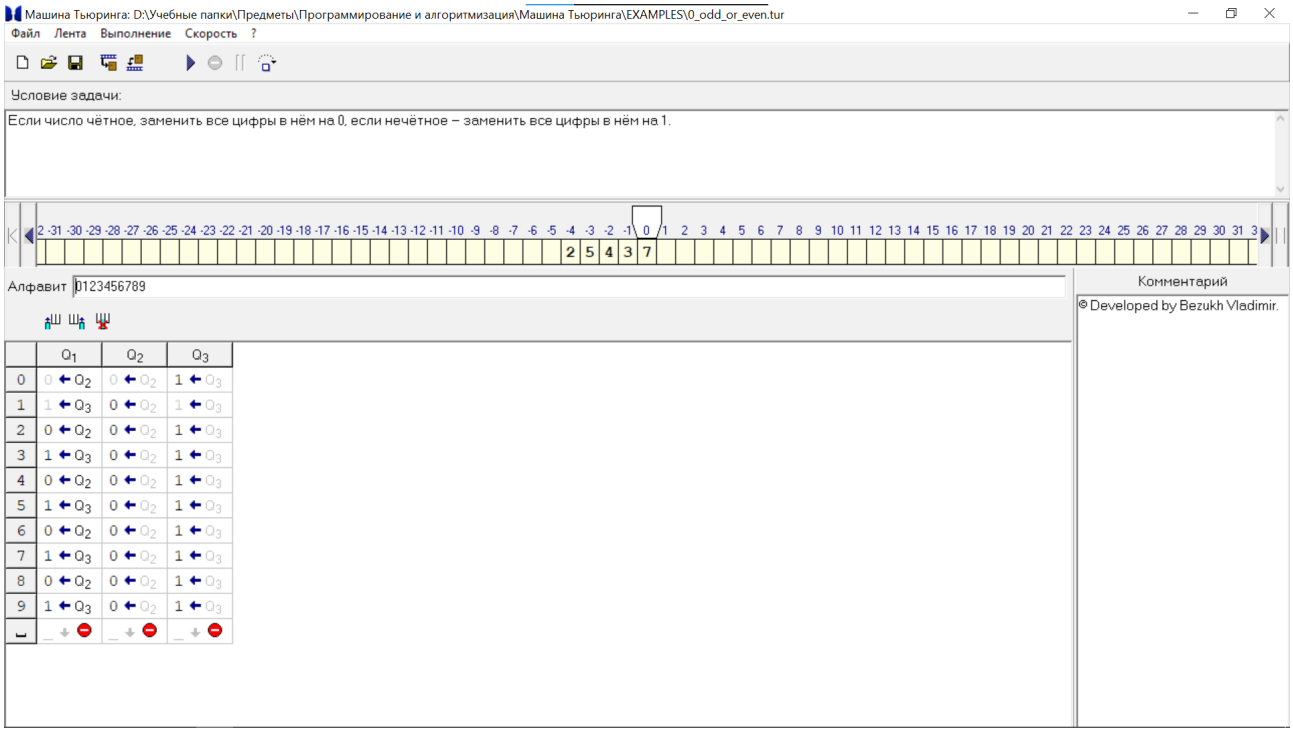

Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.

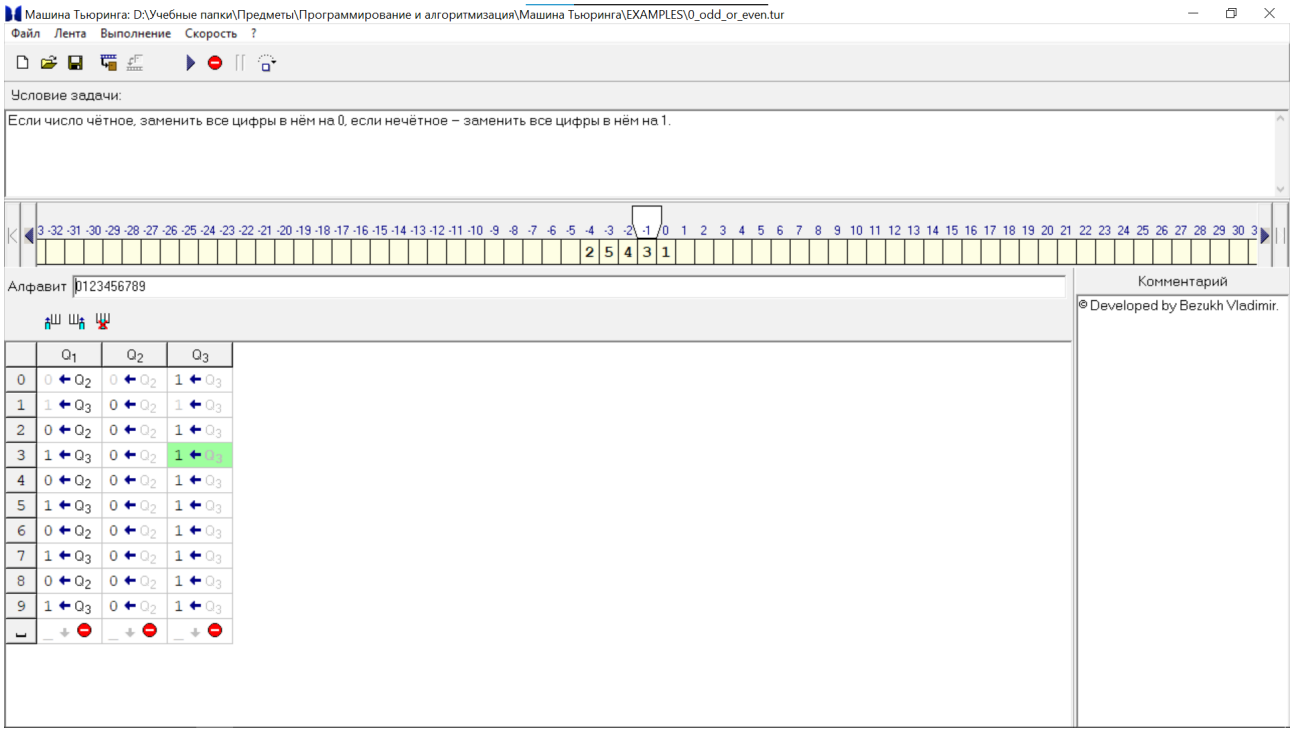

Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.

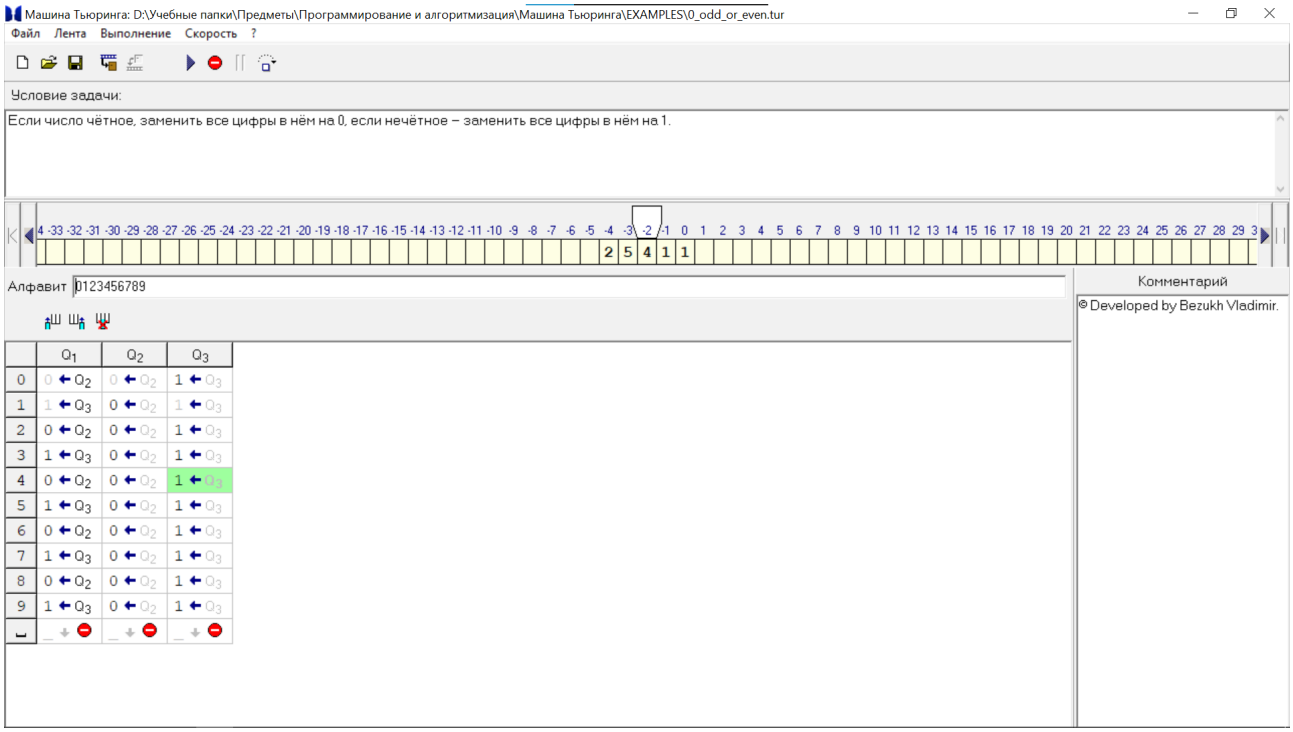

Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_dec_plus_four.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Прибавить к любому целому десятичному числу 4. Работает для всех целых чисел от «минус бесконечности» до «плюс бесконечности».
Алфавит -0123456789
Комментарий
© Developed by Bezukh Vladimir.
Информация
Выполнение программы завершено.
OK

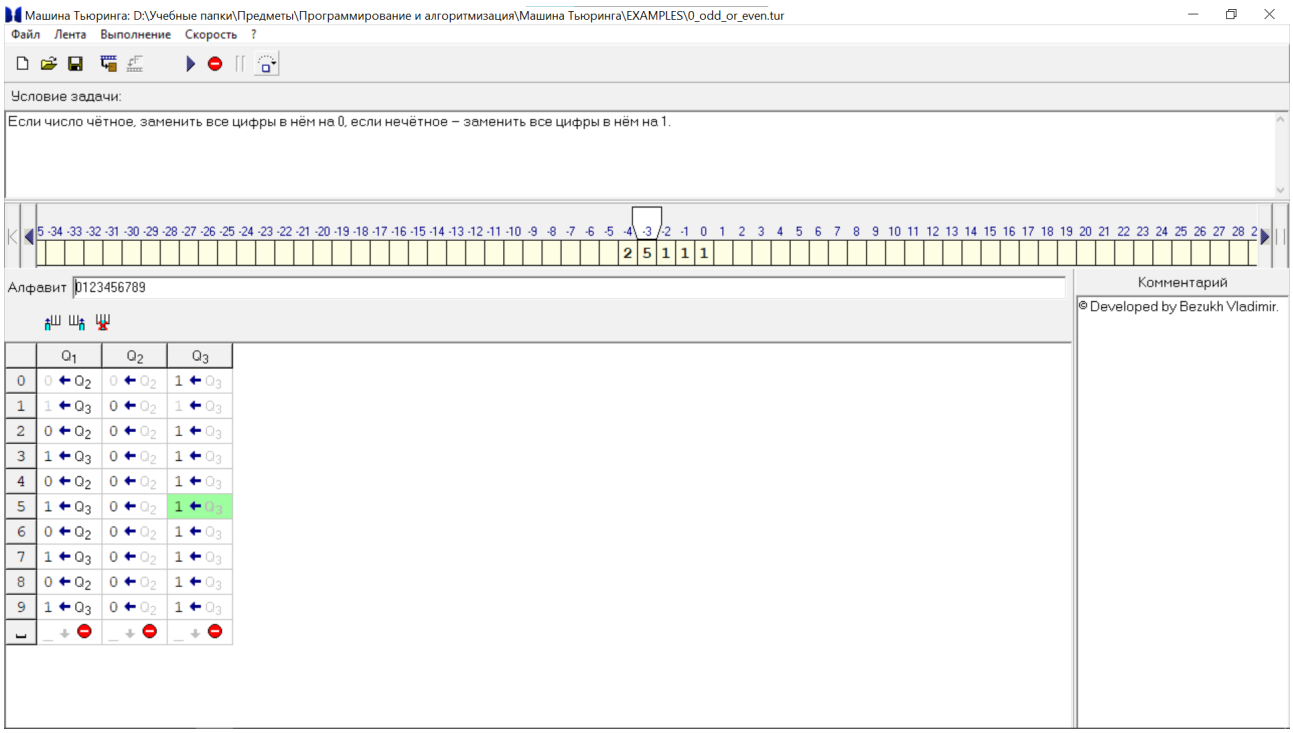
Второе задание:

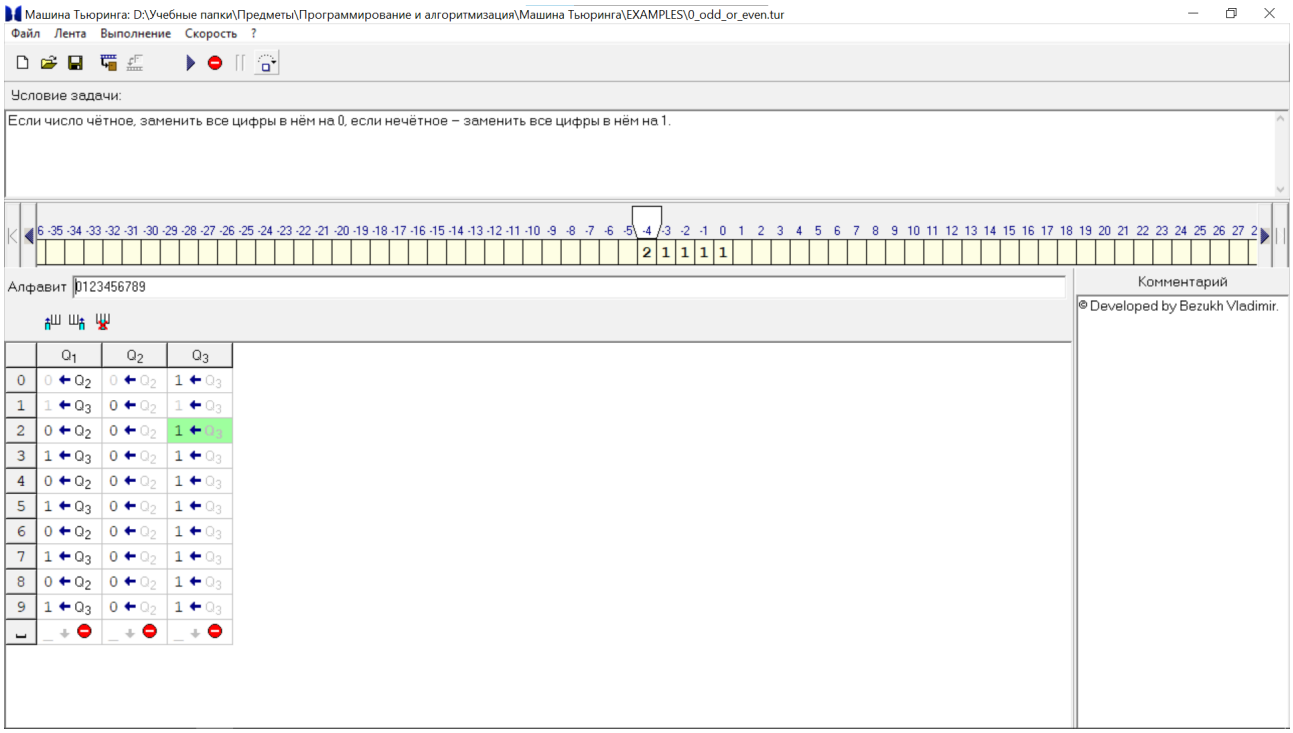
Постановка задачи указана на скриншоте. Алгоритм реализован только для положительных целых чисел, но после небольших доработок его можно применять и для отрицательных целых чисел. Определить чётность/нечётность числа можно по крайнему правому символу. Дальнейшие пояснения не требуются, т.к. очевидны из нижеприведенной реализации. Полная конфигурация машины для эмулятора Константина Полякова в файле формата .tur доступна в репозитории данного учебного проекта.

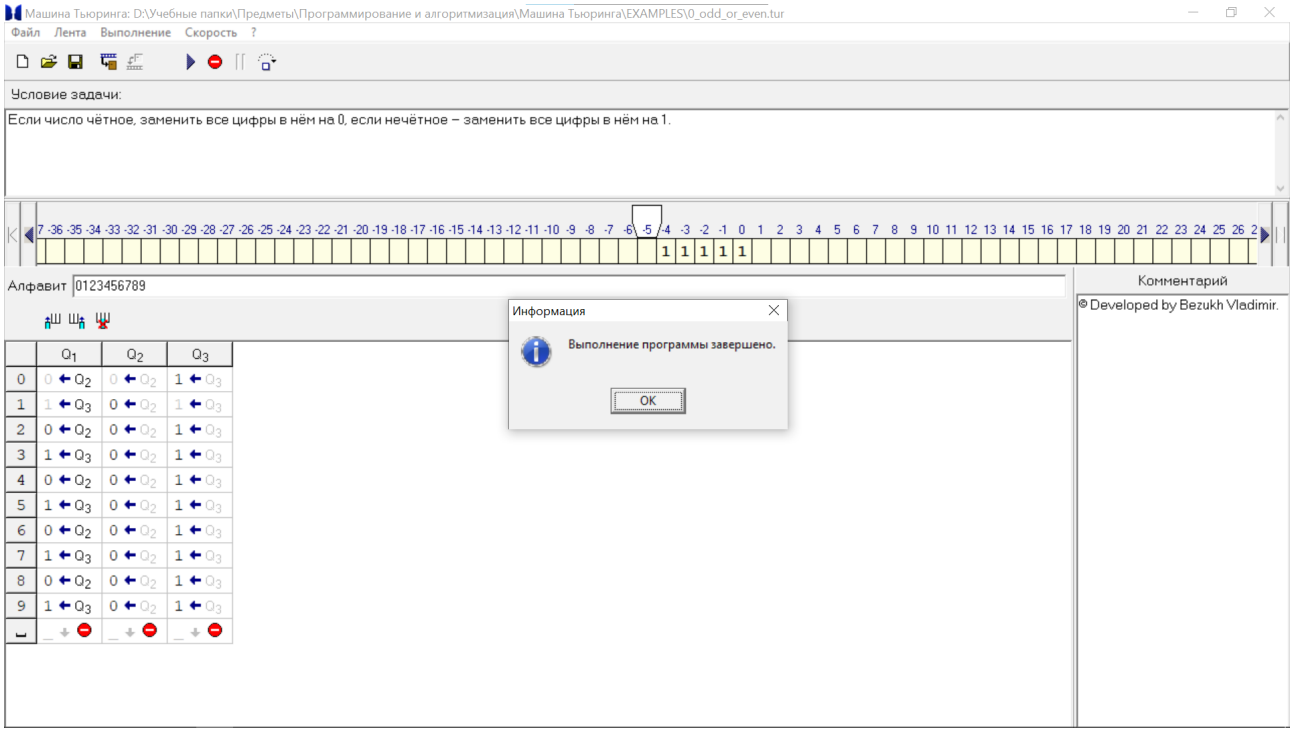





Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_odd_or_even.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Если число чётное, заменить все цифры в нём на 0, если нечётное – заменить все цифры в нём на 1.
Алфавит 0123456789
Комментарий
© Developed by Bezukh Vladimir.
Q1 Q2 Q3

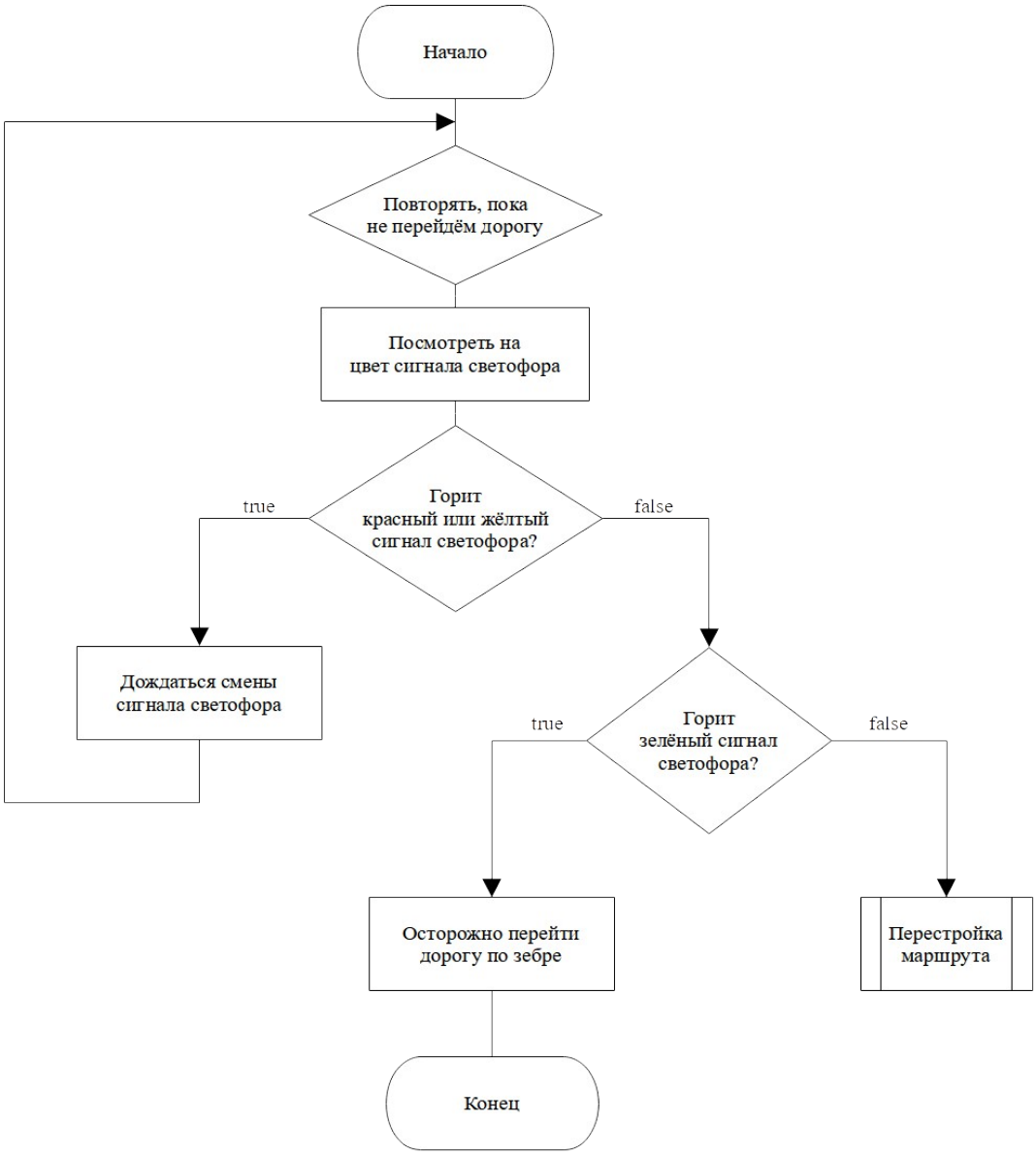

Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_odd_or_even.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Если число чётное, заменить все цифры в нём на 0, если нечётное – заменить все цифры в нём на 1.
Алфавит 0123456789
Комментарий
© Developed by Bezukh Vladimir.
Q1 Q2 Q3


Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_odd_or_even.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Если число чётное, заменить все цифры в нём на 0, если нечётное – заменить все цифры в нём на 1.
Алфавит 0123456789
Комментарий
© Developed by Bezukh Vladimir.


Машина Тьюринга: D:\Учебные папки\Предметы\Программирование и алгоритмизация\Машина Тьюринга\EXAMPLES\0_odd_or_even.tur
Файл Лента Выполнение Скорость ?
Условие задачи:
Если число чётное, заменить все цифры в нём на 0, если нечётное – заменить все цифры в нём на 1.
Алфавит 0123456789
Комментарий
© Developed by Bezukh Vladimir.
Информация
Выполнение программы завершено.
OK

Часть 3: дополнительные задания

Алгоритм перемещения через дорогу по регулируемому светофором переходу:



Дополнительное задание по НАМ:

Постановка задачи и её решение указаны на скриншоте. Пояснения к коду не требуются, т.к. используется эмулятор для НАМ из первой части отчёта.

```
void fourthNormalMarkovAlgorithmTask()
{
    /*
    * Дано:
    * A = {a, b}.
    * Преобразовать слово P так, чтобы все символы «a» оказались слева, а все символы «b» — справа.
    *
    * Решение:
    * ba → ab
    */

    const vector<MarkovRule> MARKOV_RULES = { MarkovRule("ba", false, "ab") };

    cout << "Введите строку для четвёртого задания (A = {a, b}): ";
    normalMarkovAlgorithmTaskInterface(MARKOV_RULES);
}
```

```
Консоль отладки Microsoft Visual Studio
Введите строку для четвёртого задания (A = {a, b}): bbbbbaaaaa

Строка после применения правила №1:     bbbbabaaaa              (1 шаг)
Строка после применения правила №1:     bbbabbaaaa              (2 шаг)
Строка после применения правила №1:     bbabbbaaaa              (3 шаг)
Строка после применения правила №1:     babbbbaaaa              (4 шаг)
Строка после применения правила №1:     abbbbbaaaa              (5 шаг)
Строка после применения правила №1:     abbbbabaaa              (6 шаг)
Строка после применения правила №1:     abbbabbaaa              (7 шаг)
Строка после применения правила №1:     abbabbbaaa              (8 шаг)
Строка после применения правила №1:     ababbbbaaa              (9 шаг)
Строка после применения правила №1:     aabbbbbaaa              (10 шаг)
Строка после применения правила №1:     aabbbbabaa              (11 шаг)
Строка после применения правила №1:     aabbbabbaa              (12 шаг)
Строка после применения правила №1:     aabbabbbaa              (13 шаг)
Строка после применения правила №1:     aababbbbaa              (14 шаг)
Строка после применения правила №1:     aaabbbbbaa              (15 шаг)
Строка после применения правила №1:     aaabbbbaba              (16 шаг)
Строка после применения правила №1:     aaabbbabba              (17 шаг)
Строка после применения правила №1:     aaabbabbba              (18 шаг)
Строка после применения правила №1:     aaababbbba              (19 шаг)
Строка после применения правила №1:     aaaabbbbba              (20 шаг)
Строка после применения правила №1:     aaaabbbbab              (21 шаг)
Строка после применения правила №1:     aaaabbbabb              (22 шаг)
Строка после применения правила №1:     aaaabbabbb              (23 шаг)
Строка после применения правила №1:     aaaababbbb              (24 шаг)
Строка после применения правила №1:     aaaaabbbbb              (25 шаг)
Строка после применения правила №1:     aaaaabbbbb              (26 шаг)

Результирующая строка: aaaaabbbbb
```