Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчет**

**по курсовому проекту**

**по курсу «Прикладное программирование»**

**на тему «Симулятор односторонней машины Тьюринга»**

Выполнила студентка гр. 23531/2 Васильева В.В.

(подпись)

Преподаватель Сидорина Т. Л.

(подпись)

“ ” 2018 г.

Санкт-Петербург

2018

Оглавление

[1. Техническое задание 3](#_Toc517775664)

[2. Метод решения 5](#_Toc517775665)

[3. Листинг программы 5](#_Toc517775666)

[4. Makefile 6](#_Toc517775667)

[5. Коды ошибок 7](#_Toc517775668)

[6. Программа и методы испытаний 8](#_Toc517775669)

[**Приложение 1** 21](#_Toc517775670)

[**Приложение 2** 22](#_Toc517775671)

[**Приложение 3** 24](#_Toc517775672)

[**Приложение 4** 29](#_Toc517775673)

[**Приложение 5** 31](#_Toc517775674)

# 1. Техническое задание

В данном курсовом проекте будет реализован симулятор односторонней машины Тьюринга. Программный язык реализации - С.

Отличие от стандартной машины Тьюринга заключается в том, что в процессе вычисления головка машины никогда не сдвигается левее начальной ячейки (т.е. всегда находится в ячейках с положительными номерами). В случае попытки выхода за левую границу в ходе выполнения программы, будет выводиться ошибка.

В качестве алфавита будут использоваться любые символы.

qn - n - ое состояние машины;

R - сдвиг головки вправо;

L - сдвиг головки влево;

H - головка остается на месте;

S - завершение программы.

Входные данные будут считываться из двух входных файлов.

В первом файле должны быть записаны команды переходов состояний в виде:

-aqn-bqkD , где a и b - соответсвенно символы, хранящиеся в ячейке и записывающиеся в нее;

qn и qk -состояния машины (нумерация состояний начинается с 1, в начальный момент времени машина находится в состоянии q1);

D - направление перемещения головки (одно из: L, R, H, S).

Во втором файле должно быть записано начальное состояние ленты и положение головки в виде:

\_\_\_\_\_v\_\_\_\_

1210392910

Если ввести более одного положения головки, будет учитываться только первое встретившиеся.

Полученный результат будет записан в файл вывода.

Будет предусмотрена возможность отладки написанных программ. Флагами будет указываться выполняется программа (-a) или отлаживается (-o). В отладке будут реализованы следующие возможности: выполнить один шаг (s), прервать выполнение программы (b), продолжить выполнение программы в обычном режиме (f). Так же пользователь может указать флаг (-p) для вывода в консоль всех исходных данных перед началом выполнения программы.

В выходном файле может выводиться начальное состояние ленты, а затем распечатан вывод программы в виде:

команда

лента после выполнения этой команды

Вид вывода в консоли совпадает с видом вывода в выходной файл.

Если в ходе выполнения программы выполняется более 1000 шагов, то программа прерывает выполнение с выводом соответствующей ошибки.

Вид командной строки

Turing.exe myFile.txt myFile2.txt outFile.txt -a/-o [-p]

Проект хранится на странице:

https://github.com/oxovu/Turing

Input1

-0q1-1q2L

-1q2-0q1L

-2q2-0q2S

Input2

\_\_\_v\_

20101

Output

Tape

\_\_\_v\_

20101

Running command

0q1-1q2L

Tape

\_\_v\_\_

20111

Running command

1q2-0q1L

Tape

\_v\_\_\_

20011

Running command

0q1-1q2L

Tape

v\_\_\_\_

21011

Running command

2q2-0q2S

Tape

v\_\_\_\_

01011

# 2. Метод решения

Программа состоит из 5 файлов.

Программа запускается из файла run.c.  В этом файле проверяется корректность введенных в командную строку данных. Если при запуске программы в командной строке указанно только название программы, на экран выведется краткое описание программы и пример входных и выходных данных. Затем вызывается метод input из файла input.c, который считывает все данные из файлов и записывает их: создается матрица смежности, хранящая состояния и переходы, состоящая из структур, объявленных в input.h; два вспомогательных массива, хранящие все состояния и символы, чтобы определять индексы матрицы смежности; массив символов на ленте; положение головки ленты. Программа имеет возможность задавать предполагаемые размеры матрицы и ленты. Если они будут слишком малы, то автоматически увеличатся. Все массивы создаются динамически. В этом же файле проверяется вызван ли режим debug и вызывается метод из файла step.c. В этом методе осуществляется переход состояний и запись на ленту. Если в ходе выполнения программы цикл внутри метода будет выполнен 1000 раз, программа прекратит выполнение. В файле output.c записаны методы, необходимые для вывода результата программы в консоль и файл вывода.

# 3. Листинг программы

3.1 input.h

См. Приложение 1

3.2 run.c

См. Приложение 2

3.3 input.c

См. Приложение 3

3.4 step.c

См. Приложение 4

3.5 output.c

См. Приложение 5

# 4. “Makefile”

gcc -std=c11 -pedantic -Wall -Wextra -c -o run.o run.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o input.o input.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o step.o step.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o output.o output.c

gcc -o Turing.exe run.o input.o step.o output.o

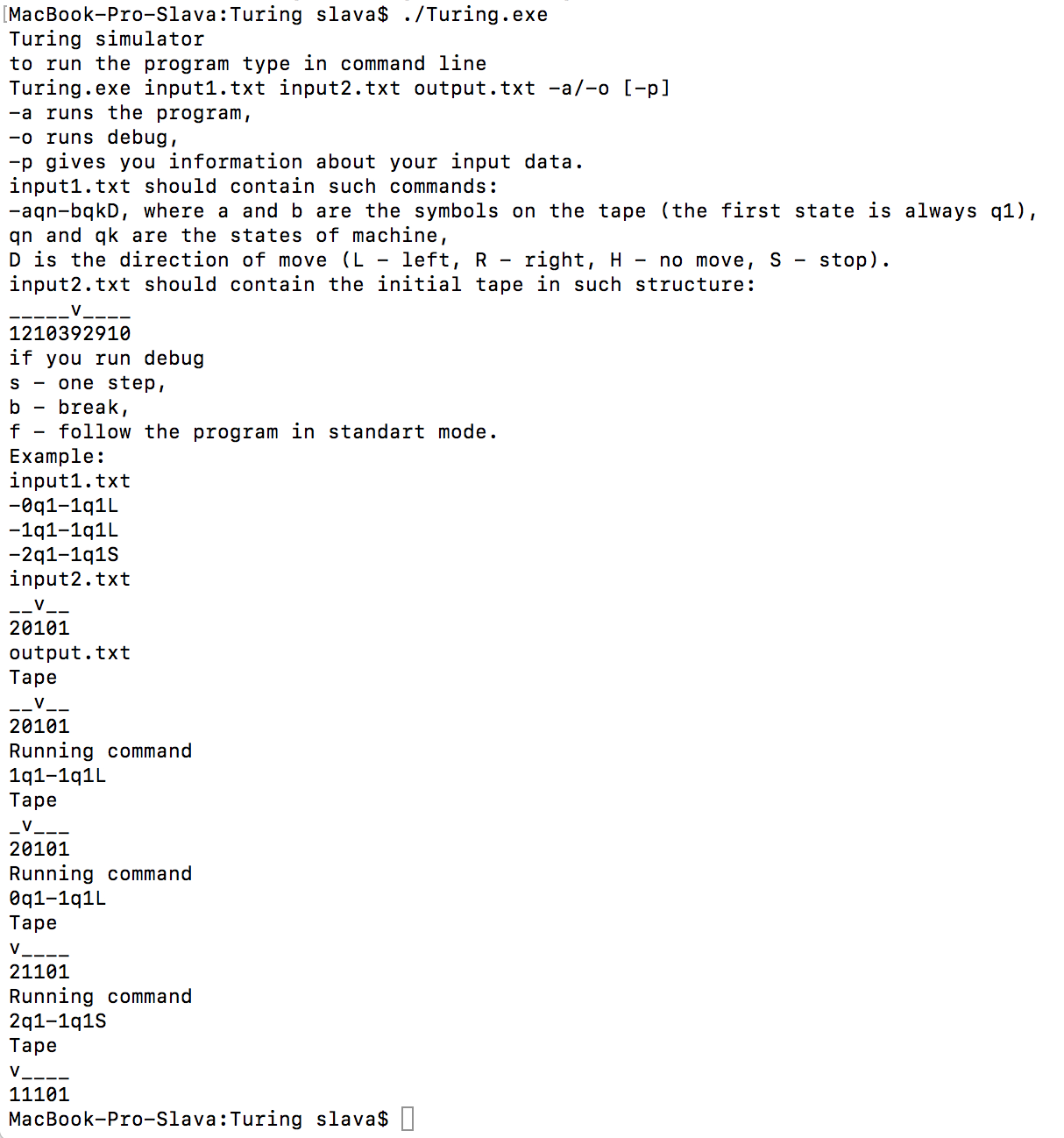
./Turing.exe myFile myFile2 outFile -a

# 5. Коды ошибок

|  |  |  |
| --- | --- | --- |
| Код ошибки | Сообщение | Действие |
| 100 | Can not find first input file | Прерывание программы |
| 101 | Wrong number of arguments | Прерывание программы |
| 102 | Memory allocation error | Прерывание программы |
| 103 | Wrong number of input chars | Прерывание программы |
| 104 | Can not find states in line | Прерывание программы |
| 106 | Wrong move command in line | Прерывание программы |
| 107 | No input data | Прерывание программы |
| 108 | Wrong input char | Прерывание программы |
| 109 | Can not find second input file | Прерывание программы |
| 200 | Head index out of bounds | Прерывание программы |
| 201 | Runtime error | Прерывание программы |
| 202 | Can not find the head | Прерывание программы |
| 203 | Don't have command for the state | Прерывание программы |

# 6. Программа и методы испытаний

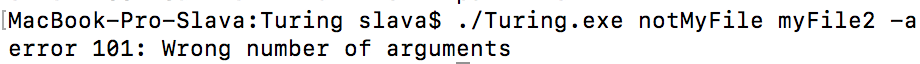
6.1 Запуск программы без аргументов



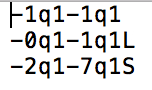
6.2 Введено название несуществующего входного файла (ошибка с кодом 100):

../../Снимок%20экрана%202018-06-26%20в%2015.06.44.png

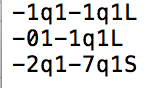
6.4 Введено неверное количество аргументов (ошибка с кодом 101):



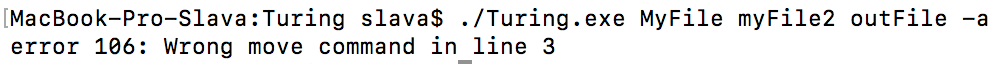
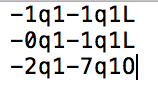
6.5 Неправильное количество симовлов в строке первого входного файла (ошибка с кодом 103):

../../Снимок%20экрана%202018-06-26%20в%2015.10.06.png

6.6 Неправильно заданные состояния (ошибка с кодом 104):

../../Снимок%20экрана%202018-06-26%20в%2015.11.25.png

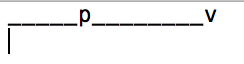
6.7 Неправильная команда сдвига головки (ошибка с кодом 106):



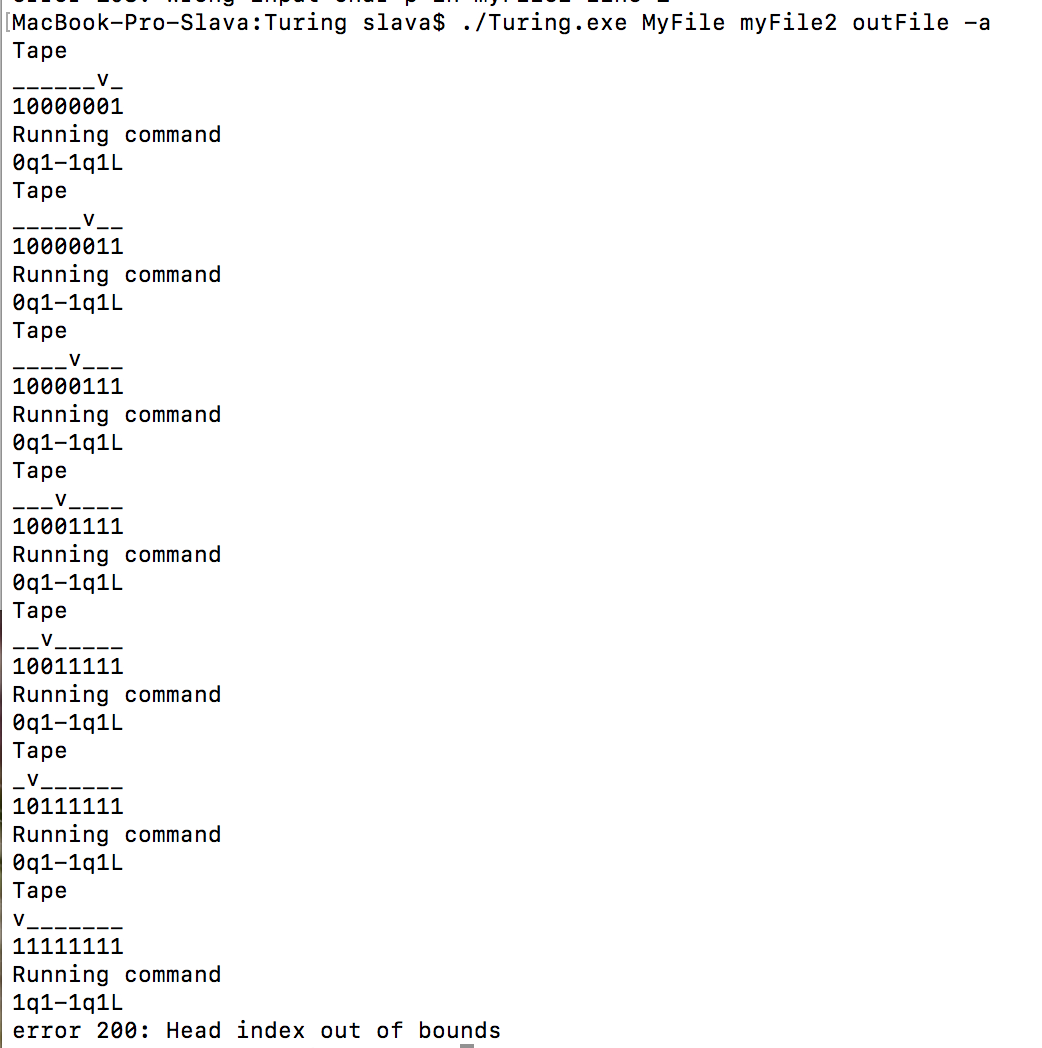
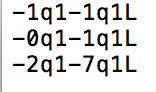
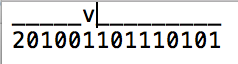
6.8 Подается пустой второй входной файл (ошибка с кодом 107):

../../Снимок%20экрана%202018-06-26%20в%2015.12.44.png

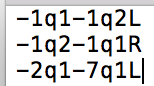
6.9 Недопустимый символ (ошибка с кодом 108):

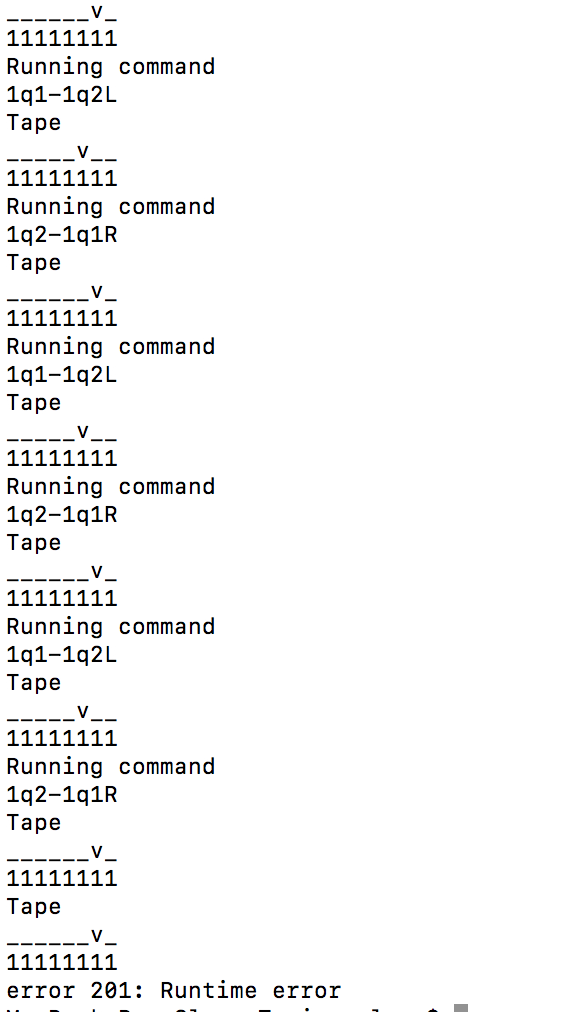
../../Снимок%20экрана%202018-06-26%20в%2015.13.10.png

6.10 Выход за начало ленты (ошибка с кодом 200):



6.11 Прерывание зацикливания (ошибка с кодом 201):

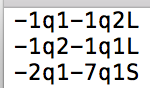
Снимок%20экрана%202018-06-26%20в%200

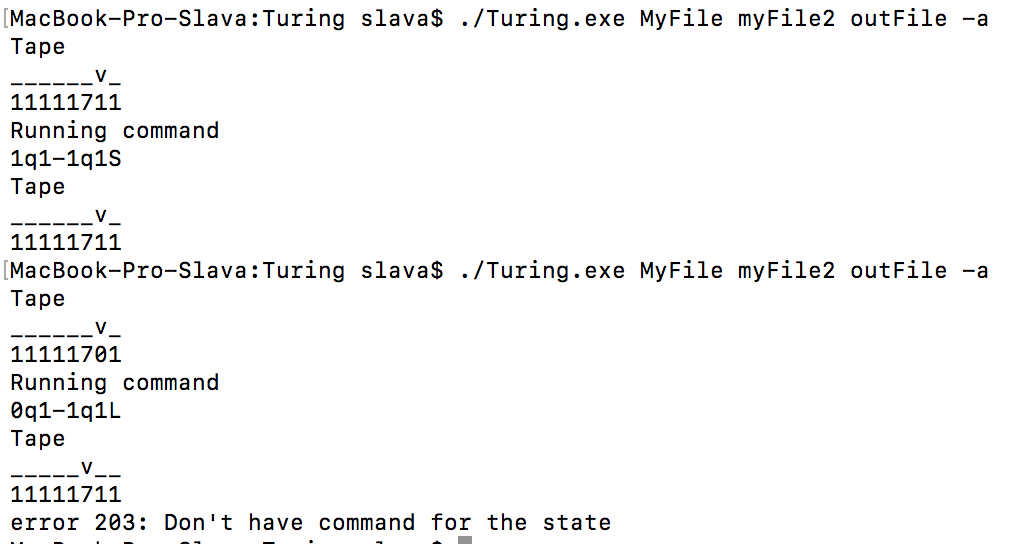


6.12 Отсутствует указатель на местоположение головки (ошибка с кодом 202):

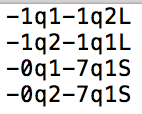
Снимок%20экрана%202018-06-26%20в%200../../Снимок%20экрана%202018-06-26%20в%2015.15.03.png

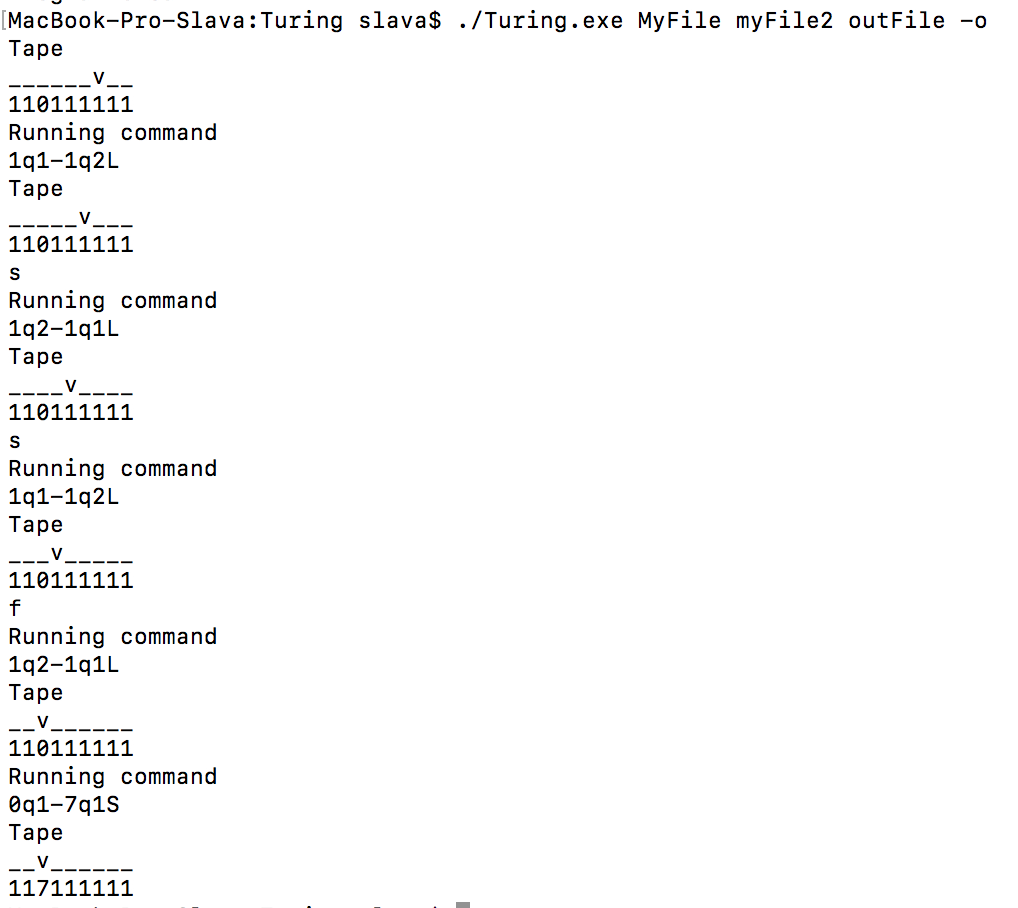
6.13 Попытка перехода в несуществующее состояние (ошибка с кодом 203):

Снимок%20экрана%202018-06-26%20в%200

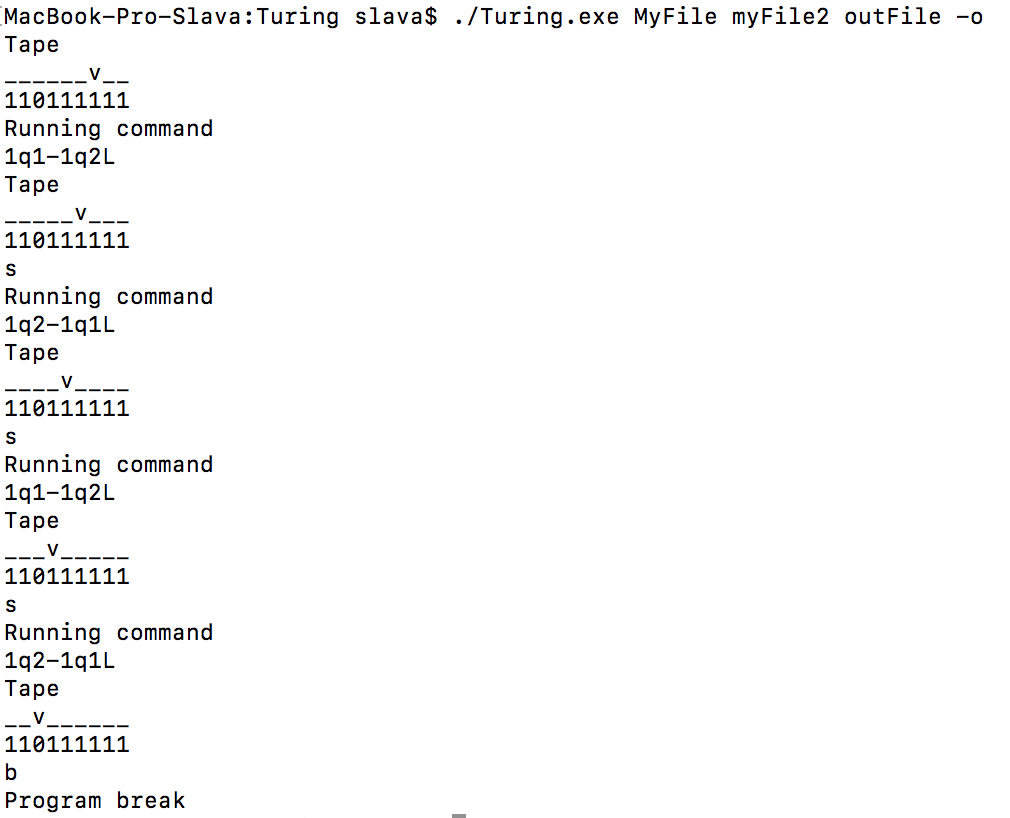


6.14 Отладка программы:

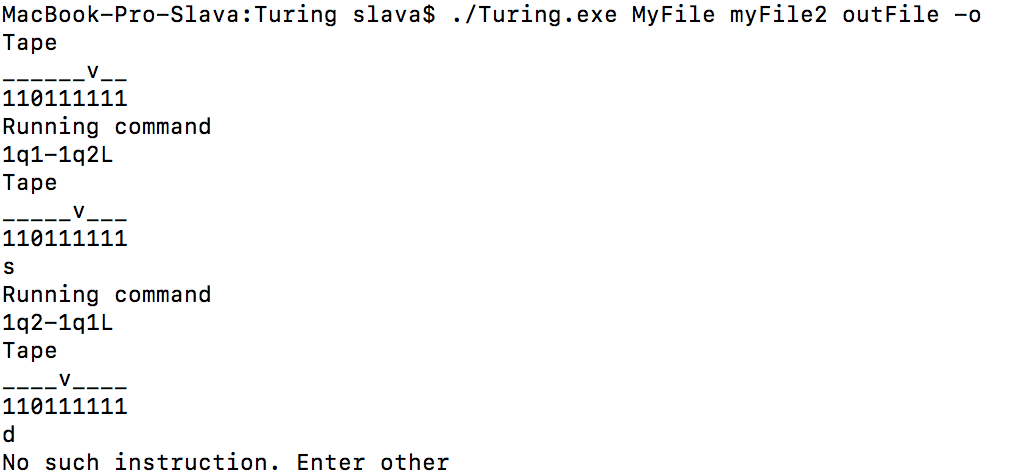




6.15 Завершение командой «b»:

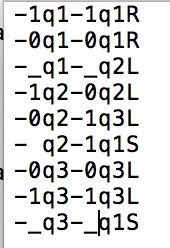
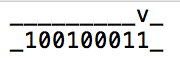


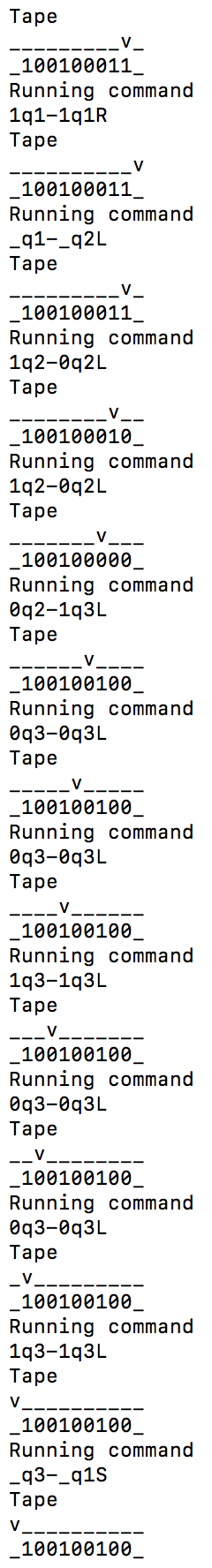
6.16 Ввод неверной команды при отдалке:



6.18 Корректная работы программы (пример 1):

Программа прибавляет единицу к двоичному числу

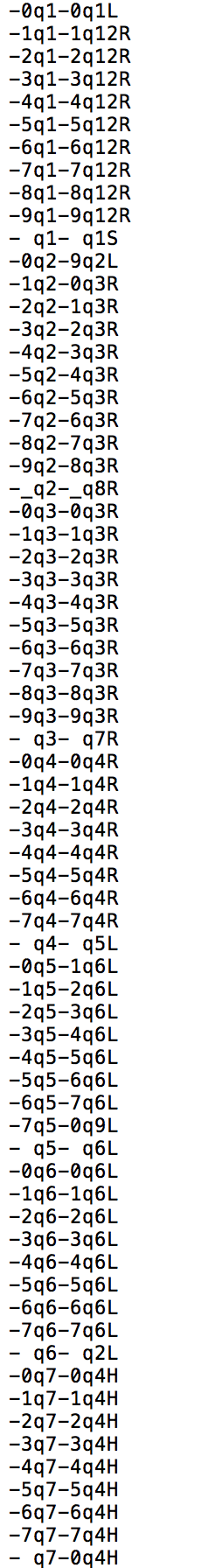


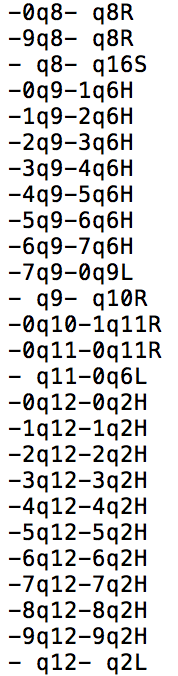


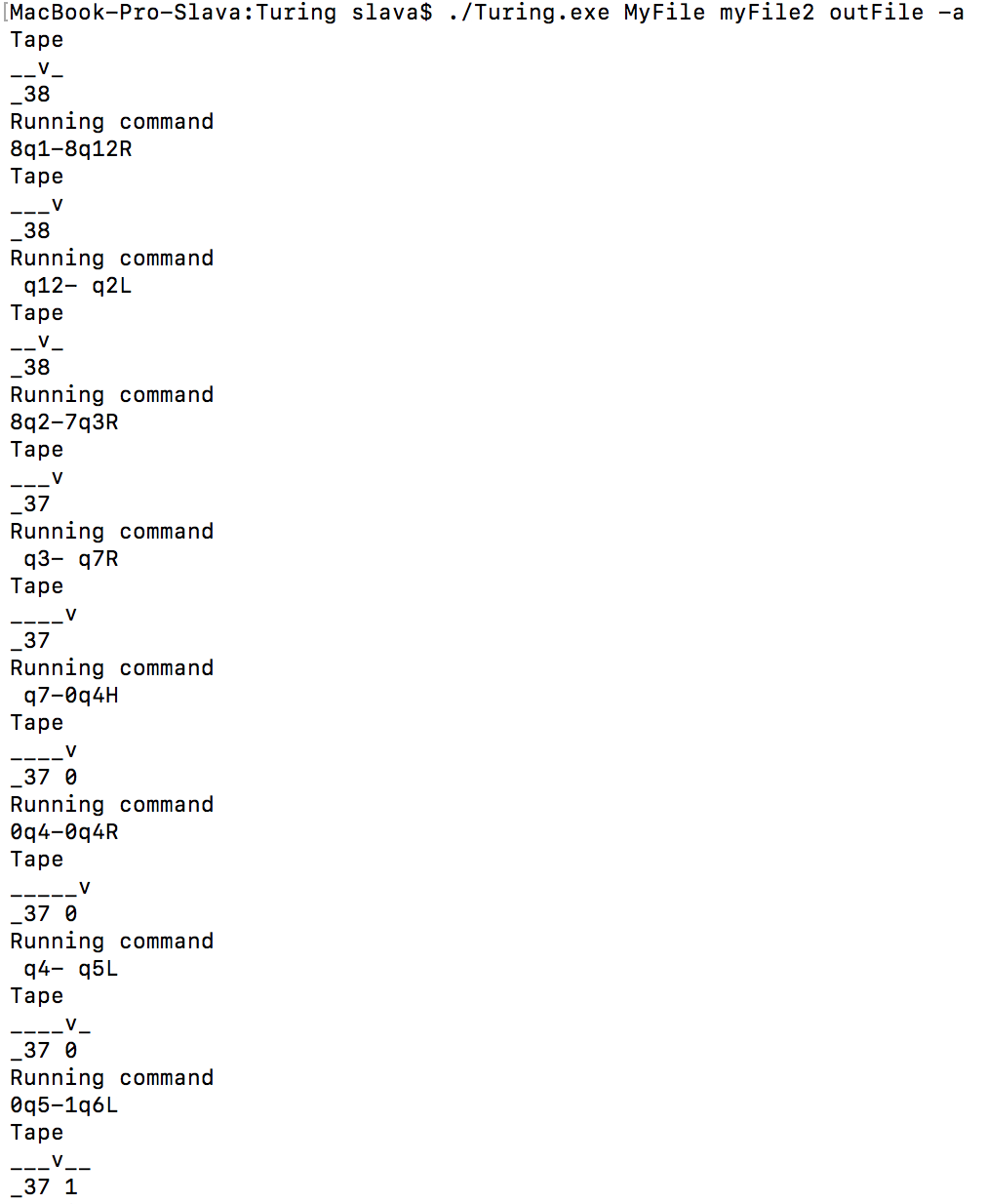
6.19 Корректная работы программы (пример 2):

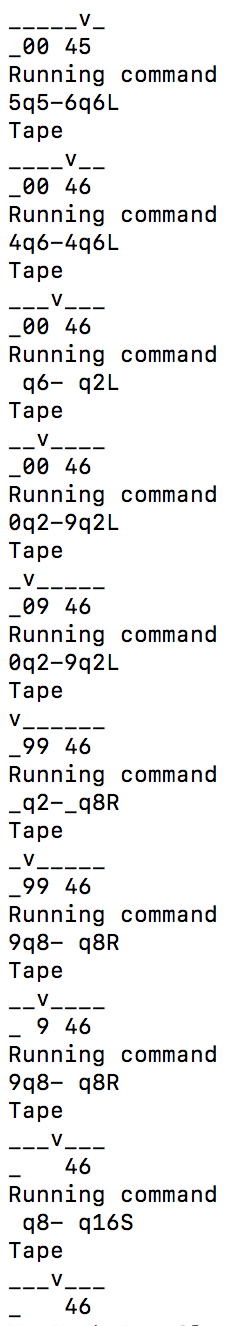
Программа переводит десятичное число в восьмеричное.

Снимок%20экрана%202018-06-26%20в%2011









# **Приложение 1**

Файл: input.h

#ifndef **TURING\_INPUT\_H**#define **TURING\_INPUT\_H**#include **<stdio.h>**#include **<stdlib.h>**#include **<string.h>**#include **<stdbool.h>  
  
struct** Command {  
 **char** newChar;  
 **int** newStateNum;  
 **char** move;  
};  
  
**int** input(FILE \*inputOne, FILE \*inputTwo, FILE \*output, **int** argc, **char** \*argv[]);  
  
**int** arrayContainsChar(**char** symbol, **char** \*arr, **int** maxSize);  
  
**int** arrayContainsInt(**int** symbol, **int** \*arr, **int** maxSize);  
  
**int** step(**int** headState, **int** lastStateNum, **char** \*tape, **char** \*symbols, **int** \*states, **struct** Command \*\*arr, **int** maxArraySize, **int** maxTapeSize,  
 **int** quit, FILE \*output, **\_Bool** flag);  
  
**void** printTape(**char** \*tape, **int** maxTapeSize, FILE \*output);  
  
**void** printHead(**int** headState, **int** maxTapeSize, FILE \*output);  
  
**void** printCommands(**struct** Command \*\*arr, **int** maxSize, FILE \*output);  
  
**void** printCommand(**struct** Command command, **char** lastChar, **int** lastStateNum, FILE \*output);  
  
#endif *//TURING\_INPUT\_H*

# **Приложение 2**

Файл: run.c

#include **"input.h"  
  
void** inf() {  
 printf(**"Turing simulator\nto run the program type in command line\nTuring.exe input1.txt input2.txt output.txt -a/-o"  
 " [-p]\n-a runs the program,\n-o runs debug,\n-p gives you information about your input data."  
 "\ninput1.txt should contain such commands:\n-aqn-bqkD, where a and b are the symbols on the tape (the"  
 " first state is always q1),\nqn"  
 " and qk are the states of machine,\nD is the direction of move (L - left, R - right, H - no move,"  
 " S - stop).\ninput2.txt should contain the initial tape in such structure:\n\_\_\_\_\_v\_\_\_\_\n1210392910\nif"  
 " you run debug\ns - one step,\nb - break,\nf - follow the program in standart mode.\nExample:"  
 "\ninput1.txt\n-0q1-1q1L\n-1q1-1q1L\n-2q1-1q1S\ninput2.txt\n\_\_v\_\_\n20101\noutput.txt\nTape\n\_\_v\_\_\n20101"  
 "\nRunning command\n1q1-1q1L\nTape\n\_v\_\_\_\n20101\nRunning command\n0q1-1q1L\nTape\nv\_\_\_\_\n21101"  
 "\nRunning command\n2q1-1q1S\nTape\nv\_\_\_\_\n11101\n"**);  
}  
  
  
**int** main(**int** argc, **char** \*argv[]) {  
 FILE \*inputOne;  
 FILE \*inputTwo;  
 FILE \*output;  
  
 **if** (argc == 1) {  
 inf(); *// информация о программе* exit(0);  
 }  
  
 **if** ((argc == 5) && (strcmp(argv[4], **"-a"**) == 0 || strcmp(argv[4], **"-o"**) == 0) ||  
 ((argc == 6) && (strcmp(argv[4], **"-a"**) == 0 || strcmp(argv[4], **"-o"**) == 0) && (strcmp(argv[5], **"-p"**) == 0))) {  
  
 inputOne = fopen(argv[1], **"rt"**);  
  
 **if** (inputOne == **NULL**) {  
 printf(**"Can not find file %s\n"**, argv[1]);  
 exit(100);  
 }  
  
 inputTwo = fopen(argv[2], **"rt"**);  
 **if** (inputTwo == **NULL**) {  
 printf(**"Can not find file %s\n"**, argv[2]);  
 exit(100);  
 }  
  
 output = fopen(argv[3], **"wr"**);  
 **if** (output == **NULL**) {  
 printf(**"Can not find file %s"**, argv[3]);  
 }  
 }  
  
 **if** (argc != 5) {  
 **if** (argc != 6 || (argc == 6 && (strcmp(argv[5], **"-p"**) != 0))) {  
 printf(**"Wrong number of arguments\n"**);  
 exit(101);  
 }  
 }  
  
 input(inputOne, inputTwo, output, argc, argv);  
  
 exit(0);  
}  
  
  
**int** arrayContainsChar(**char** symbol, **char** \*arr, **int** maxSize) { *//возвращает индекс символа или -1 если символ не найден* **for** (**int** i = 0; i < maxSize; ++i) {  
 **if** (arr[i] == symbol) **return** i;  
 }  
 **return** -1;  
}  
  
**int** arrayContainsInt(**int** symbol, **int** \*arr, **int** maxSize) { *//возвращает индекс числа или -1 если число не найдено* **for** (**int** i = 0; i < maxSize; ++i) {  
 **if** (arr[i] == symbol) **return** i;  
 }  
 **return** -1;  
}

# **Приложение 3**

Файл: input.c

#include **"input.h"***// в этом примере используется  
// myFile myFile2 outFile -a*#define **ARRAY\_SIZE** 7 *//начальная длина матрицы смежности*#define **TAPE\_SIZE** 4 *//начальная длина ленты***int** input(FILE \*inputOne, FILE \*inputTwo, FILE \*output, **int** argc, **char** \*argv[]) {  
  
 **\_Bool** flag = **false**;  
  
 *//переменные для чтения из файла состояний* **char** lastChar; *// символ на ленте* **char** lastState;  
 **int** lastStateNum; *// состояние в котором находится лента* **char** dash;  
 **char** newChar; *// символ, который нужно записать* **char** newState;  
 **int** newStateNum; *// со //переменные для чтения из файла состоянийстояние, в которе должна перейти лента* **char** move; *//* **int** maxArraySize = **ARRAY\_SIZE**; *//размер матрицы смежности* **int** \*states; *//вспомогательный массив, хранящий состояния* states = calloc(**ARRAY\_SIZE**, **sizeof**(states));  
 **if** (states == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 **char** \*symbols; *//вспомагательный массив, хранящий символы ленты* symbols = calloc(**ARRAY\_SIZE**, **sizeof**(symbols));  
 **if** (symbols == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
  
 **int** i = 0; *//счетчик для заполнения марицы смежности* **int** statesCount = 0; *//текущий индекс массива states* **int** symbolsCount = 0; *//текущий индекс массива symbols* **struct** Command blankCommand = {**'\0'**, 0, **'\0'**}; *//пустая команда* **struct** Command \*\*command; *//таблица смежности* command = calloc(**ARRAY\_SIZE**, **sizeof**(**struct** Command \*));  
  
 **if** (command == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
  
 **for** (**int** j = 0; j < **ARRAY\_SIZE**; ++j) {  
 command[j] = calloc(**ARRAY\_SIZE**, **sizeof**(**struct** Command));  
 **if** (command[j] == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 }  
  
  
 **int** ret = 0; *//сечтчик fscanf* **int** line = 0;  
  
 **while** (**true**) { *//чтение файла с командами* line++;  
  
 ret = fscanf(inputOne, **"%c%c%c%d%c%c%c%d%c\n"**, &dash, &lastChar, &lastState, &lastStateNum, &dash, &newChar,  
 &newState, &newStateNum, &move);  
 **if** (ret == **EOF**) **break**;  
  
 **if** (ret != 9) {  
 printf(**"Wrong number of input chars\n"**);  
 exit(103);  
 }  
  
 **if** (lastState != **'q'** || newState != **'q'**) {  
 printf(**"Can not find states in line %d\n"**, line);  
 exit(104);  
 }  
  
 **if** (move != **'L'** && move != **'R'** && move != **'H'** && move != **'S'**) {  
 printf(**"Wrong move command in line %d\n"**, line);  
 exit(106);  
 }  
  
 **if** (arrayContainsChar(lastChar, symbols, maxArraySize) == -1) {  
 symbols[symbolsCount] = lastChar;  
 symbolsCount++;  
 }  
  
 **if** (arrayContainsInt(lastStateNum, states, maxArraySize) == -1) {  
 states[statesCount] = lastStateNum;  
 statesCount++;  
 }  
  
 i++;  
 **if** (i == maxArraySize) { *//увеличение матрицы при необходимости* maxArraySize = maxArraySize \* 2;  
 states = realloc(states, maxArraySize \* **sizeof**(**int**));  
 **if** (states == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 symbols = realloc(symbols, maxArraySize \* **sizeof**(**char**));  
 **if** (symbols == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 command = realloc(command, maxArraySize \* **sizeof**(**struct** Command \*));  
 **if** (command == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
  
 **for** (**int** j = 0; j < maxArraySize / 2; ++j) {  
 command[j] = realloc(command[j], maxArraySize \* **sizeof**(**struct** Command));  
 **if** (command[j] == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 **for** (**int** k = maxArraySize / 2; k < maxArraySize; ++k) {  
 command[j][k] = blankCommand;  
 }  
 }  
  
 **for** (**int** k = maxArraySize / 2; k < maxArraySize; ++k) {  
 command[k] = calloc(maxArraySize, **sizeof**(**struct** Command));  
 **if** (command[k] == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 }  
 }  
  
 *//запись команд в матрицу смежности* command[arrayContainsChar(lastChar, symbols, maxArraySize)][arrayContainsInt(lastStateNum, states,  
 maxArraySize)].newChar = newChar;  
 command[arrayContainsChar(lastChar, symbols, maxArraySize)][arrayContainsInt(lastStateNum, states,  
 maxArraySize)].newStateNum = newStateNum;  
 command[arrayContainsChar(lastChar, symbols, maxArraySize)][arrayContainsInt(lastStateNum, states,  
 maxArraySize)].move = move;  
 }  
  
 fclose(inputOne);  
  
 **if** (argc == 6 && (strcmp(argv[5], **"-p"**) == 0)) { *// печать матрицы смежности* printCommands(command, maxArraySize, output);  
 }  
  
 **char** \*head; *//переменные для чтения файла с лентой* head = calloc(**TAPE\_SIZE**, **sizeof**(**char**));  
 **if** (head == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
  
 **int** headState = -1; *//индекс положения головки машины* **char** \*tape;  
 tape = calloc(**TAPE\_SIZE**, **sizeof**(**char**));  
 **if** (tape == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
  
 **int** maxTapeSize = **TAPE\_SIZE**;  
  
 ret = fscanf(inputTwo, **" %s"**, head);  
  
 **if** (ret == -1) {  
 printf(**"No input data in %s line 1\n"**, argv[2]);  
 exit(107);  
 }  
  
 **while** (head[maxTapeSize] != **'\0'**) { *//увеличение размера ленты при необходимости* maxTapeSize++;  
 head = realloc(head, maxTapeSize \* **sizeof**(**int**));  
 **if** (head == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 tape = realloc(tape, maxTapeSize \* **sizeof**(**char**));  
 **if** (tape == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 **for** (**int** j = maxTapeSize - 1; j < maxTapeSize; ++j) {  
 tape[j] = **'\0'**;  
 }  
 }  
 fseek(inputTwo, 0, **SEEK\_SET**);  
 ret = fscanf(inputTwo, **" %s"**, head);  
  
 **if** (ret == -1) {  
 printf(**"No input data in %s line 1\n"**, argv[2]);  
 exit(107);  
 }  
  
 **for** (**int** l = 0; l < maxTapeSize; ++l) {  
 **if** (head[l] != **'\_'** && head[l] != **'v'** && head[l] != **'\0'**) {  
 printf(**"Wrong input char %c in %s line 1\n"**, head[l], argv[2]);  
 exit(108);  
 }  
 **if** (head[l] == **'v'**) {  
 headState = l;  
 **break**;  
 }  
 }  
  
 **if** (headState == -1) {  
 printf(**"Can not find the head\n"**);  
 exit(202);  
 }  
  
 fscanf(inputTwo, **"\n"**);  
  
 **for** (**int** n = 0; n < maxTapeSize; ++n) {  
 tape[n] = **' '**;  
 }  
  
 **for** (**int** n = 0; n < maxTapeSize; ++n) {  
 fscanf(inputTwo, **"%c"**, &tape[n]);  
 }  
  
 fclose(inputTwo);  
  
 **if** (strcmp(argv[4], **"-o"**) == 0) { *// вызван ли режим debug* flag = **true**;  
 }  
  
 printHead(headState, maxTapeSize, output);  
 printTape(tape, maxTapeSize, output);  
  
 **int** quit = 0; *//количество шагов* step(headState, 1, tape, symbols, states, command, maxArraySize, maxTapeSize, quit, output, flag);  
  
 free(head);  
 free(tape);  
 free(states);  
 free(symbols);  
 **for** (**int** m = 0; m < maxArraySize; ++m) {  
 free(command[m]);  
 }  
 fclose(output);  
}

# **Приложение 4**

Файл: step.c

#include **"input.h"  
  
int**step(**int** headState, **int** lastStateNum, **char** \*tape, **char** \*symbols, **int** \*states, **struct** Command \*\*arr, **int** maxArraySize,  
 **int** maxTapeSize, **int** quit, FILE \*output, **\_Bool** flag) {  
 **while** (quit != 1000) {  
 quit++;  
 **char** lastChar = tape[headState];  
 **int** i = arrayContainsChar(lastChar, symbols, maxArraySize); *// поиск индексов* **int** j = arrayContainsInt(lastStateNum, states, maxArraySize);  
 **if** (i == -1 || j == -1) {  
 printf(**"Don't have command for the state\n"**);  
 exit(203);  
 }  
 printCommand(arr[i][j], lastChar, lastStateNum, output);  
 lastStateNum = arr[i][j].newStateNum;  
 tape[headState] = arr[i][j].newChar;  
 **switch** (arr[i][j].move) {  
 **case 'L'**:  
 headState--;  
 **break**;  
 **case 'R'**:  
 headState++;  
 **break**;  
 }  
 **if** (headState == maxTapeSize) {  
 maxTapeSize++;  
 tape = realloc(tape, maxTapeSize \* **sizeof**(**char**));  
 **if** (tape == **NULL**) {  
 printf(**"Memory allocation error\n"**);  
 exit(102);  
 }  
 tape[maxTapeSize - 1] = **' '**;  
 }  
 **if** (headState < 0) {  
 printf(**"Head index out of bounds\n"**);  
 exit(200);  
 }  
 **if** (arr[i][j].move == **'S'**) {  
 **break**;  
 }  
 printHead(headState, maxTapeSize, output);  
 printTape(tape, maxTapeSize, output);  
 **if** (flag) {  
 **char** comand[2]; *//сюда будет записана команда* **while** (1) {  
 scanf(**" %2s"**, comand);  
 fflush(**stdin**); *//очистка буфера* **if** (!strcmp(comand, **"b"**)) {  
 printf(**"Program break\n"**);  
 fclose(output);  
 exit(0);  
 }  
 **if** (!strcmp(comand, **"s"**)) {  
 **break**;  
 }  
 **if** (!strcmp(comand, **"f"**)) {  
 flag = **false**;  
 **break**;  
 } **else** printf(**"No such instruction. Enter other\n"**);  
 }  
 }  
 }  
  
 printHead(headState, maxTapeSize, output);  
 printTape(tape, maxTapeSize, output);  
  
 **if** (quit == 1000) {  
 printf(**"Runtime error\n"**);  
 exit(201);  
 }  
 **return** headState;  
}

# **Приложение 5**

Файл: output.c

#include **"input.h"  
  
  
void** printTape(**char** \*tape, **int** maxTapeSize, FILE \*output) {  
  
 **int** i = 0;  
 **while** (tape[i] != **'\0'** && i < maxTapeSize) {  
 printf(**"%c"**, tape[i]);  
 fprintf(output, **"%c"**, tape[i]);  
 i++;  
 }  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
}  
  
**void** printHead(**int** headState, **int** maxTapeSize, FILE \*output) {  
 printf(**"Tape"**);  
 fprintf(output, **"Tape"**);  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
 **int** i = 0;  
 **while** (i != maxTapeSize) {  
 **if** (i == headState){  
 printf(**"v"**);  
 fprintf(output, **"v"**);  
 }  
 **if** (i != headState){  
 printf(**"\_"**);  
 fprintf(output, **"\_"**);  
 }  
 i++;  
 }  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
}  
  
**void** printCommands(**struct** Command \*\*arr, **int** maxArraySize, FILE \*output){  
 printf(**"Command matrix"**);  
 fprintf(output, **"Command matrix"**);  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
 **for** (**int** k = 0; k < maxArraySize; ++k) {  
 **for** (**int** j = 0; j < maxArraySize; ++j) {  
 printf(**"%c%d%c "**, arr[k][j].newChar, arr[k][j].newStateNum, arr[k][j].move);  
 fprintf(output, **"%c%d%c "**, arr[k][j].newChar, arr[k][j].newStateNum, arr[k][j].move);  
 }  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
 }  
}  
  
**void** printCommand(**struct** Command command, **char** lastChar, **int** lastStateNum, FILE \*output){  
 printf(**"Running command"**);  
 fprintf(output, **"Running command"**);  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
 printf(**"%cq%d-%cq%d%c"**, lastChar, lastStateNum, command.newChar, command.newStateNum, command.move);  
 fprintf(output, **"%cq%d-%cq%d%c"**, lastChar, lastStateNum, command.newChar, command.newStateNum, command.move);  
 printf(**"\n"**);  
 fprintf(output, **"\n"**);  
}