Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

**Кафедра «Компьютерные системы и программные технологии»**

**КУРСОВОЙ ПРОЕКТ**

**Симулятор машин Поста**

по дисциплине «Прикладное программирование»

Выполнила

студентка гр.33501/1 Е.Ю.Сидорина

Руководитель К.А.Гагарский

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Санкт-Петербургский государственный политехнический университет

**ЗАДАНИЕ**

**НА ВЫПолнение курсовоГО ПРОЕКТА**

студентке группы 23531/1 Сидориной Елене Юрьевне

***1. Тема проекта (работы):*** Симулятор машин Поста

***2. Срок сдачи законченного проекта (работы)*** 31.05.18 г.

***3. Исходные данные к проекту (работе)***:

Описание машины, содержимое входной ленты задаются во входных текстовых файлах (разных), результат работы машины выводится в текстовый файл.

Необходимо обеспечить возможность имитации работы машины до ее останова или в течение заданного числа шагов.

Необходимо обеспечить возможность отладки машины (вывод промежуточных конфигураций, возобновление работы машины).

Допускается введение разумных ограничений на число инструкций (миллионы). Размер ленты должен быть ограничен только доступным симулятору объемом памяти.

Разработка кода и документации должна вестись в репозитории git.

Разработанное ПО должно собираться с помощью команды make компилятором gcc (mingw) с опциями -std=c11 -pedantic -Wall -Wextra без предупреждений компилятора.

***4. Содержание пояснительной записки***: введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

***Дата получения задания***: «5» апреля 2018 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А.Гагарский

*(подпись)*

Задание приняла к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Ю.Сидорина

*(подпись студента)*

05.04.2018г.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . …………….. | 4 |
| 1. Методика решения…………………………………………………............... | 5 |
| 2. Код программы, реализующей симулятор……………………................... | 7 |
| 2.1. post.h ……………………………………………………………………. | 7 |
| 2.2. main.c ………………………………………………………………….... | 7 |
| 2.3. menu.c ……………………….………………………………………….. | 9 |
| 2.4. tape.c ……………………………………………………………………. | 10 |
| 2.5. comand.c ………………………………………………………………… | 14 |
| 2.6. run.c ……………………………………………………………………... | 17 |
| 3. Сборка разработанного ПО с помощью команды make …..……………. | 20 |
| 4. Методика испытаний……………………………………………………….. | 20 |
| 4.1. Командная строка………………………………………………………. | 20 |
| 4.2. Лента…………………………………………………………………….. | 21 |
| 4.3. Команды………………………………………………………………… | 23 |
| 4.4. Печать…………………………………………………………………… | 27 |
| 4.5. Мгновенное выполнение программы……..………………………….. | 30 |
| 4.6. Пошаговое выполнение программы…………………………………. | 32 |
| 4.7. Восстановление ленты………………………………………………… | 35 |
| 4.8. Пример нормальной работы программы…………………………….. | 35 |
| Заключение .................................................................................................................. | 39 |
| Список использованных источников  ........................................................................ | 40 |

**ВВЕДЕНИЕ**

В данном курсовом проекте мною будет написана программа, являющаяся симулятором машины Поста, что соответствует моему индивидуальному заданию по прикладному программированию.

В ней будут реализованы основные опции машины Поста, такие как загрузка ленты и команд из соответствующих входных файлов, печать ленты и команд на экран, пошаговое и мгновенное выполнение программы, содержащейся в файле с командами, а также опция восстановления ленты.

**1. МЕТОДИКА РЕШЕНИЯ**

Имена всех файлов ввода/вывода задаются в командной строке при запуске программы. Вид командной строки, откуда программа «берет» имена двух входных и одного выходного файлов в формате .txt:

post.exe ftape.txt fcom.txt out.txt

а) ftape.txt - файл, хранящий положение головки на ленте и состояние ленты в следующем формате:

\_\_\_\_\_\_\_\_\_\_

       v

1110001001

\_\_\_\_\_\_\_\_\_\_\_\_

, где первая строка – это положение головки (перед символом «v», обозначающим положение головки, должно стоять ровно столько пробелов, сколько ячеек перед той, на которой головка), а вторая строка – это состояние ленты (где «0» - ячейка не помечена, а «1» - ячейка помечена).

б) fcom.txt - файл, содержащий список команд программы, которую должен будет выполнить симулятор, в следующем формате:

\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

2. ? 1,3

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_

, где первая цифра до точки – это номер команды, а далее расположена сама команда.

в) out.txt - файл вывода, содержащий пошаговую работу выполненной программы, создается по желанию пользователя.

Набор допустимых команд:

< c1 – сдвиг влево и переход к команде с номером c1 (номера команд задаются перед командами и идут подряд: 1. , 2. , 3. и т.д.)

> c1 – сдвиг вправо и переход к команде с номером c1

1 c1 – отметка текущей ячейки ленты и переход к команде с номером c1

0 c1 – снятие отметки с текущей ячейки и переход к команде с номером c1

? c1,c2 – если текущая ячейка не отмечена - переход к команде с номером c1, если отмечена - переход к команде с номером c2

. – остановка программы.

Данные ленты хранятся в булевском массиве bool tape[], где каждый элемент массива – это ячейка (помеченная – true или нет – false).

Список команд хранится в массиве структур struct COMMAND commands[].

struct COMMAND

{

int number, // номер команды

condition1, // с1

condition2; // с2

char command; // команда ( '<', '>', '1', '0', '?', '.' )

char \*commandString; // полный вид команды "1. < 2"

};

Функциональность файлов программы:

post.h – содержит необходимые макросы, прототипы функций и т.п.;

main.c – главная функция программы;

menu.c – функция для работы меню выбора действий пользователя;

tape.c – функции для работы с лентой (загрузка, печать, …);

comand.c – функции для работы с командами (загрузка, печать, …);

run.c – функция для выполнения самой работы машины Поста.

Работа с пользователем осуществляется посредством меню, представленного на

Рис 1.1:

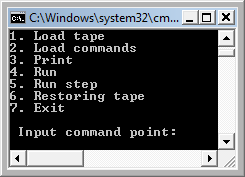


Рис 1.1 Меню работы симулятора

При работе в режиме отладки (п.5. Run step) предусмотрены три возможных действия:

1. Сделать следующий шаг - <*Enter*>

2. Прервать отладку – ввести команду *stop*

3. Выполнить программу до конца – ввести команду *run*

При запуске работы машины (п.4 Run и п.5. Run step) у пользователя спрашивается:

1) нужна ли запись в файл вывода (если нет – только на экран, да – на экран и в файл).

2) максимальное количество выполнения команд (защита от зацикливания). По умолчанию = 100.

**2. Код программы, реализующей симулятор**

**2.1. post.h**

#ifndef POSTMACHINE\_POST\_H

#define POSTMACHINE\_POST\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define TAPE\_SIZE 50

#define TAPE\_RESIZE 20

#define COMMAND\_SIZE 50

#define COMMAND\_RESIZE 20

#define ENTER 10

int menu(int lastPoint);

void clearTape(bool \*\*tape, int tapeSize, int \*tapePointer, bool \*\*copyTapeValue);

bool loadTape(char \*fTape, bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue);

void mark(bool \*tape,int tapePointer);

void unmark(bool \*tape, int tapePointer);

void right(bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue);

void left(bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue);

void printTape(FILE \*wherePrint, bool \*tape, int tapeSize, int tapePointer, bool ifLoadTape);

void copyTape(bool \*what, bool \*whereTo, int size);

struct COMMAND

{

int number, condition1, condition2;

char command;

char \*commandString;

};

void clearCommands(struct COMMAND \*\*commands, int commandSize,

int \*numberOfCommands);

bool loadCommands(char \*fCom,struct COMMAND \*\*commands, int \*commandSize,

int \*numberOfCommands);

void printCommandsList(struct COMMAND \*commands, int numberOfCommands,

bool ifLoadCommand);

int lennum(int num); //количество цифр в номере строки

void run(char \*fOut, bool step,

bool \*\*tape, int \*tapePointer, int \*tapeSize, bool \*\*copyTapeValue, bool ifLoadTape,

struct COMMAND \*commands, bool ifLoadCommand);

#endif //POSTMACHINE\_POST\_H

**2.2. main.c**

#include "post.h"

int main(int argc, char \*args[])

{

// menu

int pointMenu;

// tape

int tapeSize = TAPE\_SIZE;

bool \*tape = NULL;

bool \*copyTapeValue = NULL;

int tapePointer = 0;

int start\_tapePointer=0;

bool ifLoadTape = false; //лента не загружена

// command

int commandSize = COMMAND\_SIZE;

struct COMMAND \*commands=NULL;

int numberOfCommands=0;

bool ifLoadCommand = false;

if (argc != 4)

{

printf("post.exe ftape.txt fcom.txt out.txt \n");

return -1;

}

do

{

pointMenu = menu(7);

switch (pointMenu)

{

case 1: clearTape(&tape, tapeSize, &tapePointer, &copyTapeValue);

 ifLoadTape = loadTape(args[1], &tape, &tapeSize,

&tapePointer, &copyTapeValue);

 start\_tapePointer = tapePointer;

 printTape(stdout, tape, tapeSize, tapePointer, ifLoadTape );

  break;

case 2: clearCommands(&commands, commandSize,

&numberOfCommands);

 ifLoadCommand = loadCommands(args[2], &commands,

&commandSize, &numberOfCommands);

 printCommandsList(commands,numberOfCommands,

ifLoadCommand);

 break;

case 3: if(ifLoadTape)

printTape(stdout, tape, tapeSize, tapePointer, ifLoadTape);

 else

printf("Tape wasn't load \n \n");

 if(ifLoadCommand)

printCommandsList(commands,numberOfCommands,

ifLoadCommand);

 else

printf("Commands wasn't load \n \n");

 break;

case 4: run(args[3], false, &tape, &tapePointer, &tapeSize,

&copyTapeValue, ifLoadTape, commands, ifLoadCommand);

 break;

case 5: run(args[3], true, &tape, &tapePointer, &tapeSize,

&copyTapeValue, ifLoadTape, commands, ifLoadCommand);

 break;

case 6: if(!ifLoadTape)

 {

printf("\n You need to load tape!!\n\n");

break;

 }

 copyTape( copyTapeValue, tape, tapeSize);

 tapePointer = start\_tapePointer;

 printTape(stdout, tape, tapeSize, tapePointer, ifLoadTape);

 break;

case 7: break;

}

} while (pointMenu != 7);

if( tape != NULL )

free(tape);

if( copyTapeValue != NULL )

free(copyTapeValue);

if( commands != NULL )

free(commands);

printf ("\n See you later :3 \n \n \n");

return 0;

}

**2.3. menu.c**

#include "post.h"

int menu(int lastPoint)

{

int point, ret;

printf( "1. Load tape \n"

"2. Load commands \n"

"3. Print \n"

"4. Run \n"

"5. Run step \n"

"6. Restoring tape \n"

"7. Exit \n");

while (1)

{

printf ("\n Input command point: ");

ret = scanf("%d", &point);

if (ret == 1 && (point >= 1 && point <= lastPoint))

break;

fflush(stdin);

printf("Error. Point must be in [1..%d] interval. Please, try again \n" ,lastPoint);

}

return point;

}

**2.4. tape.c**

#include "post.h"

/\*

v

1110001000

\*/

//очистка ленты

void clearTape(bool \*\*tape, int tapeSize, int \*tapePointer, bool \*\*copyTapeValue)

{

if(\*tape != NULL)

free(\*tape);

\*tape = (bool \*)calloc(tapeSize, sizeof(bool));

if(\*tape == NULL)

{

printf("Not enough memory for tape\n");

exit(100);

}

if(\*copyTapeValue != NULL)

free(\*copyTapeValue);

\*copyTapeValue = (bool \*)calloc(tapeSize, sizeof(bool) );

if( \*copyTapeValue == NULL )

{

printf("Not enough memory for copyTapeValue\n");

exit(101);

}

\*tapePointer = (tapeSize)/2;

}

//загрузка ленты

bool loadTape(char \*fTape, bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue)

{

char tmp[81];

int lenghtOfString, ret;

char n; // '0' or '1'

FILE \*in;

int v = -1;

bool ifLoadTape = false; //лента не загружена

int oldsize = \*tapeSize;

in = fopen(fTape, "rt");

if (in == NULL)

{

printf("File %s not found \n", fTape);

exit(-2);

}

fgets(tmp, 80, in);

lenghtOfString = strlen(tmp);

for(int i=0; i<lenghtOfString; i++)

{

if (tmp[i] == 'v')

{

v = i;

break;

}

if (tmp[i] != ' ')

{

printf("Error in data file %s/ There are may be only spases or v \n", fTape);

fclose(in);

return ifLoadTape;

}

}//end for

if (v == -1)

{

printf("Error in data file %s/ There are must be v \n", fTape);

fclose(in);

return ifLoadTape;

}

do

{

ret = fscanf(in, "%c", &n);

if (ret == EOF) //конец файла

{

break;

}

if (ret != 1 || (n != '0' && n != '1'))

{

printf("Error in data file %s/ There are must be only 0 and 1 \n", fTape);

fclose(in);

return ifLoadTape;

}

if (n=='1')

{

mark(\*tape, \*tapePointer);

}

else

{

unmark(\*tape, \*tapePointer);

}

right(tape, tapeSize, tapePointer, copyTapeValue);

} while (!feof(in));

fclose(in);

\*tapePointer = oldsize/2 + v;

ifLoadTape = true;

copyTape(\*tape, \*copyTapeValue, \*tapeSize);

return ifLoadTape;

}

//поставить метку

void mark(bool \*tape, int tapePointer)

{

tape[tapePointer] = true;

}

//снять метку

void unmark(bool \*tape, int tapePointer)

{

tape[tapePointer] = false;

}

//сдвиг вправо

void right(bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue)

{

(\*tapePointer)++;

if (\*tapePointer < \*tapeSize)

return;

\*tapeSize += TAPE\_RESIZE;

\*tape = (bool \*)realloc((bool \*)\*tape, \*tapeSize);

if(\*tape == NULL)

{

printf("Not enough memory for tape-right\n");

exit(102);

}

\*copyTapeValue = realloc((bool \*)\*copyTapeValue, \*tapeSize);

if(\*copyTapeValue == NULL)

{

printf("Not enough memory for copyTapeValue-right\n");

exit(103);

}

}

//сдвиг влево

void left(bool \*\*tape, int \*tapeSize, int \*tapePointer, bool \*\*copyTapeValue)

{

(\*tapePointer)--;

if (\*tapePointer >= 0)

return;

int oldSize = \*tapeSize;

bool \*oldTape = (bool \*)calloc(oldSize, sizeof(bool));

if(oldTape == NULL)

{

printf("Not enough memory for oldTape-left\n");

exit(104);

}

copyTape(\*tape, oldTape, oldSize);

\*tapeSize += TAPE\_RESIZE;

\*tape = (bool \*)realloc((bool \*)\*tape, \*tapeSize);

if(\*tape == NULL)

{

printf("Not enough memory for tape-left\n");

exit(105);

}

//копируем с конца

for (int i = oldSize-1, j = \*tapeSize-1; i>=0; i--, j--)

{

\*(\*tape+j) = \*(oldTape+i);

}

\*tapePointer = \*tapeSize-oldSize-1;

for (int i = 0; i <= \*tapePointer; i++)

{

\*(\*tape+i) = false;

}

\*copyTapeValue = realloc((bool \*)\*copyTapeValue, \*tapeSize);

if(\*copyTapeValue == NULL)

{

printf("Not enough memory for copyTapeValue-left\n");

exit(106);

}

free(oldTape);

}

void copyTape(bool \*what, bool \*whereTo, int size)

{

int i;

for (i=0; i < size; i++)

{

\*(whereTo+i) = \*(what+i);

}

}

void printTape(FILE \*wherePrint, bool \*tape, int tapeSize, int tapePointer, bool ifLoadTape)

{

int width = 70; //ширина поля печати

int delta = width/2;

int i = tapePointer - delta; //левый край печати

if (!ifLoadTape)

{

printf("\n Tape wasn't load! \n");

return;

}

fprintf(wherePrint, "%\*c \n", delta+1, 'v');

for (int k=0; k < width; k++, i++) //k - сколько символов напечатать

{

if (i<0 || i>tapeSize || tape[i] == false)

{

fprintf(wherePrint, "%c", '.');

}

else

fprintf(wherePrint, "%c", '1');

}

fprintf(wherePrint, "\n \n");

}

**2.5. comand.c**

#include "post.h"

/\*

1. <2

2. ?1,3

3. >4

4. 1 5

5. .

\*/

//очистка списка команд

void clearCommands(struct COMMAND \*\*commands, int commandSize,

int \*numberOfCommands)

{

if(\*commands != NULL)

free(\*commands);

\*commands = (struct COMMAND \*)calloc(commandSize, sizeof(struct COMMAND));

if(\*commands == NULL)

{

printf("Not enough memory for commands\n");

exit(200);

}

\*numberOfCommands = 0;

}

//загрузка команд

bool loadCommands(char \*fCom,struct COMMAND \*\*commands, int \*commandSize,

int \*numberOfCommands)

{

int ret, number, condition1, condition2, length;

int resize = COMMAND\_RESIZE;

char command;

FILE \*in;

bool ifLoadCommand = false;

const char \*commandTypes = "<>?10";

\*numberOfCommands = 0;

int len;

in = fopen(fCom, "rt");

if (in == NULL)

{

printf("File %s not found \n", fCom);

exit(-3);

}

do

{

ret = fscanf(in, " %d. %c %d", &number, &command, &condition1);

if (ret == EOF)

break;

if (number <= 0 || condition1 <= 0 )

{

printf("Error in %s file. There may be only commands from [1…]

interval \n", fCom);

fclose(in);

return ifLoadCommand;

}

(\*numberOfCommands)++;

//проверка последовательности введенных команд

if (\*numberOfCommands != number)

{

printf("Error in %s file. Command numbers must be in order \n", fCom);

fclose(in);

return ifLoadCommand;

}

if (\*numberOfCommands > \*commandSize)

{

\*commandSize += resize;

\*commands = (struct COMMAND \*)realloc(

(struct COMMAND \*)\*commands, \*commandSize);

}

(\*commands+number-1)->number = number;

(\*commands+number-1)->command = command;

(\*commands+number-1)->condition1 = condition1;

// len(number)+'.'+ ' ' + command('.') + '\0' = len(number)+4

len = lennum(number)+4;

if (ret == 2 && command == '.')

{

(\*commands+number-1)->commandString=(char \*)calloc(

len, sizeof(char));

sprintf((\*commands+number-1)->commandString, "%d. %c", number,

command);

break;

}

if (ret != 3)

{

printf("Error in %s file. Command must have a certain structure \n",

fCom);

fclose(in);

return ifLoadCommand;

}

length = strlen(commandTypes);

int i;

for (i=0; i < length; i++)

{

if (command == commandTypes[i])

break;

}

if (i == length)

{

printf("Error in %s file. Сommand does not exist \n", fCom);

fclose(in);

return ifLoadCommand;

}

len += 1 + lennum(condition1); // ' ' + len(condition1)

(\*commands+number-1)->condition2=-1;

if (command == '?')

{

ret = fscanf(in, ",%d", &condition2);

if (ret != 1 || condition2 <= 0 || condition2 == condition1)

{

printf("Error in %s file. Wrong condition2 \n", fCom);

fclose(in);

return ifLoadCommand;

}

(\*commands+number-1)->condition2 = condition2;

len+= 1+ lennum(condition2); // ',' + len(condition2)

}

(\*commands+number-1)->commandString=(char \*)calloc(len, sizeof(char));

if ((\*commands+number-1)->condition2!= -1)

sprintf((\*commands+number-1)->commandString, "%d. %c %d,%d",

number, command, condition1, condition2);

else

sprintf((\*commands+number-1)->commandString, "%d. %c %d",

number, command, condition1);

} while(!feof(in));

fclose(in);

if ((\*commands+(\*numberOfCommands)-1)->command != '.')

{

printf("Error in %s file. No stop-command - . in last command \n", fCom);

return ifLoadCommand;

}

for(int i=0; i< (\*numberOfCommands); i++)

{

if((\*commands+i)->condition1 > \*numberOfCommands ||

(\*commands+i)->condition2 > \*numberOfCommands)

{

printf("Error condition1 or condition2 in command %d \n", i+1);

return ifLoadCommand;

}

}

ifLoadCommand = true;

return ifLoadCommand;

}

void printCommandsList(struct COMMAND \*commands, int numberOfCommands,

bool ifLoadCommand)

{

if (!ifLoadCommand)

{

printf("\n Commands wasn't load \n");

return;

}

printf("\n \n");

for (int i=0; i < numberOfCommands; i++)

{

printf("%s \n", commands[i].commandString);

}

printf("\n \n");

}

int lennum(int num)

{

int col =0;

while( num )

{

col++;

num=num/10;

}

return col;

}

**2.6. run.c**

#include "post.h"

//bool step = true - пошагово, а false - сразу

void run(char \*fOut, bool step,

bool \*\*tape, int \*tapePointer, int \*tapeSize, bool \*\*copyTapeValue, bool ifLoadTape,

struct COMMAND \*commands, bool ifLoadCommand)

{

char save;

bool needFile = false;

FILE \*out;

char currentCommand; //текущая команда

int i = 0, //индекс текущей команды (номер команды - 1)

c; //или condition1, или condittion2

int maxNumberOfCommands = 100; //максимально возможное количество шагов

int currentNumberOfCommands = 0;

int ret;

int key; //код возвращаемой клавиши

char stepcommand[5]; // "stop" или "run"

if (!ifLoadTape || !ifLoadCommand)

{

printf("Error! Commands (or tape) wasn't uploaded! \n");

return;

}

printf("Save to file? [y-yes, any - no] \n");

scanf(" %c", &save);

if (save == 'y')

{

needFile = true;

out = fopen(fOut, "wt");

if (out == NULL)

{

printf("File %s can not be open \n", fOut);

exit(-4);

}

}

fflush(stdin);

printf("\n Input maxNumberOfCommands \n");

ret = scanf(" %d", &maxNumberOfCommands);

if (ret != 1)

printf("Wrong value! Default maxNumberOfCommands = 100 \n");

fflush(stdin);

if(step)

printf(" <Enter> - next step, stop - stop, run - run\n");

printTape(stdout, \*tape, \*tapeSize, \*tapePointer, ifLoadTape);

if (needFile)

printTape(out, \*tape, \*tapeSize, \*tapePointer, ifLoadTape);

while (1)

{

printf("%s \n", commands[i].commandString);

if (needFile)

fprintf(out, "%s \n", commands[i].commandString);

currentCommand = commands[i].command;

switch(currentCommand)

{

case '1': mark(\*tape, \*tapePointer);

break;

case '0': unmark(\*tape, \*tapePointer);

break;

case '<': left(tape, tapeSize, tapePointer, copyTapeValue);

break;

case '>': right(tape, tapeSize, tapePointer, copyTapeValue);

break;

}

if (currentCommand != '.' && currentCommand != '?')

{

printTape(stdout, \*tape, \*tapeSize, \*tapePointer, ifLoadTape);

if (needFile)

printTape(out, \*tape, \*tapeSize, \*tapePointer, ifLoadTape);

}

else if (currentCommand == '.')

{

printf("\n \n STOP \n \n ");

if (needFile)

fprintf(out, "\n \n STOP \n \n ");

break;

}

c = commands[i].condition1;

if (currentCommand == '?')

{

if (\*(\*tape+(\*tapePointer)) == true)

c = commands[i].condition2;

}

i = c - 1;

currentNumberOfCommands++;

if (currentNumberOfCommands > maxNumberOfCommands)

{

printf("Exceeded maxNumberOfCommands %d \n",

maxNumberOfCommands);

if (needFile)

fprintf(out, "Exceeded maxNumberOfCommands %d \n",

maxNumberOfCommands);

break;

}

if (step)

{

while(1)

{

key = getc(stdin);

if(key == ENTER)

break;

ungetc(key,stdin); //вернуть "первый" символ обратно в поток,

чтобы считать все слово

scanf(" %4s",stepcommand);

fflush(stdin);

if(!strcmp(stepcommand,"run")) // == 0

{

step = false;

break;

}

if(!strcmp(stepcommand,"stop"))

{

printf("Aborted by user \n \n \n");

if (needFile)

{

fprintf(out, "Aborted by user \n \n \n");

fclose(out);

}

return;

}

printf("error");

fflush(stdin);

}

} // end if (step)

}//end while

if (needFile)

fclose(out);

}

**3. Сборка разработанного ПО с помощью команды make**

**Makefile:**

\_\_\_\_\_\_\_\_\_\_\_\_\_

post : main.o menu.o tape.o comand.o run.o

gcc -o post main.o menu.o tape.o comand.o run.o

main.o : main.c

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

menu.o : menu.c

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

tape.o : tape.c

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

comand.o : comand.c

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

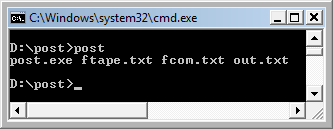
run.o : run.c

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

\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Методика испытаний**

**4.1. Командная строка**



**4.2. Лента**

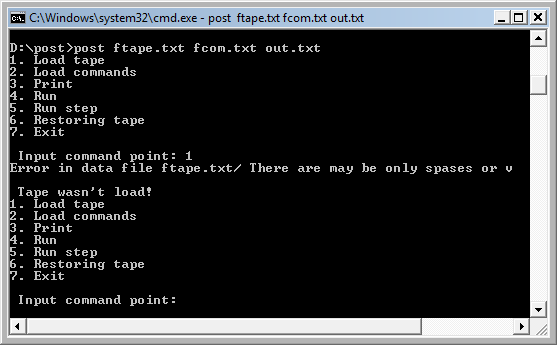
1) Отсутствие позиции головки

ftape.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1110001001

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



2) Наличие недопустимых символов в указании положения головки:

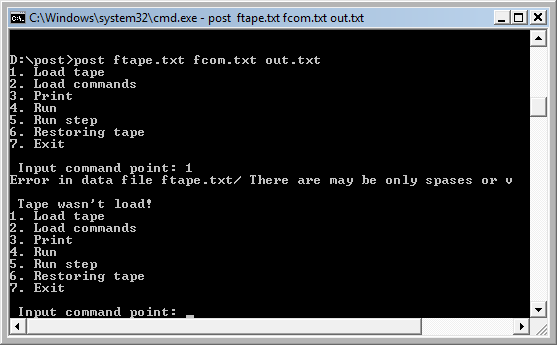
ftape.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

  x v

1110001001

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



3) Наличие недопустимых символов (0/1) в ячейках ленты: 1a10001000

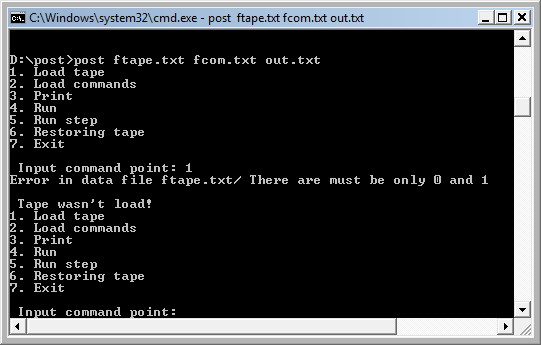
ftape.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    v

1a10001001

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



4) Нормальная загрузка ленты:

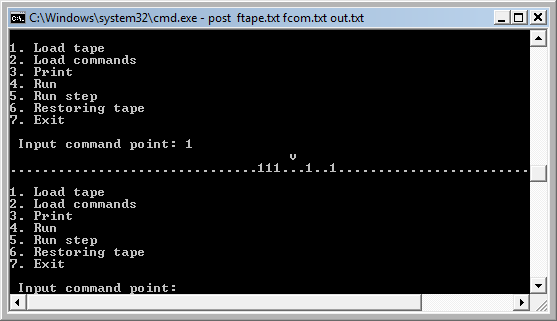
ftape.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

    v

1110001001

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**4.3. Команды**

1) Неверная строка команды: a. < 2

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a. < 2

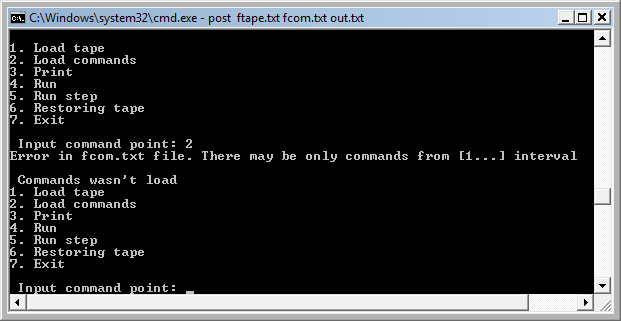
2. ? 1,3

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



2) Неверная последовательность нумерации команд: 20. ? 1,3

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

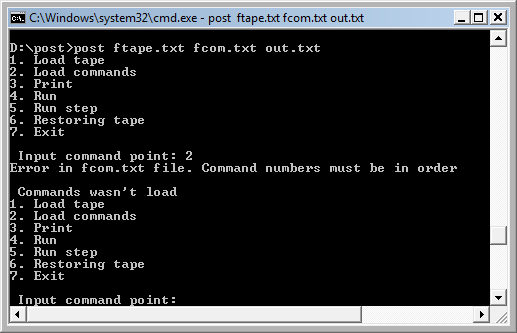
20. ? 1,3

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



3) Неверная команда: 1. x 2

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. x 2

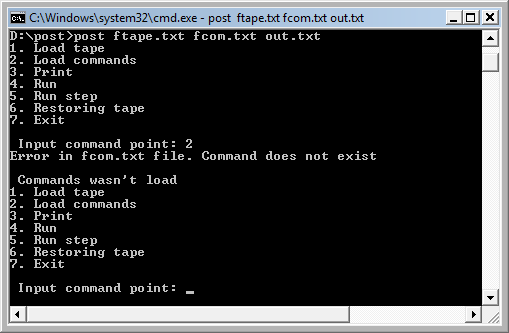
2. ? 1,3

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



4) Неверная метка перехода: 2. ? 1,1

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

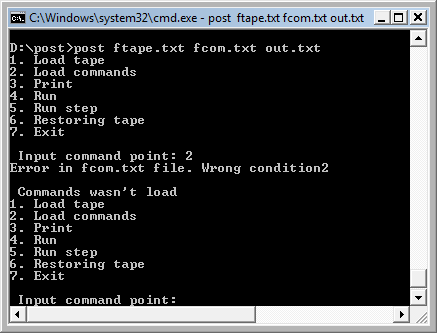
2. ? 1,1

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



5) Неверное значение метки: 1. < -2

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < -2

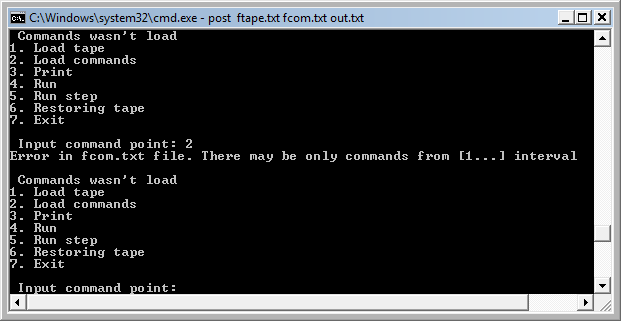
2. ? 1,3

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



6) Отсутствие второй метки в условии: 2. ? 1

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

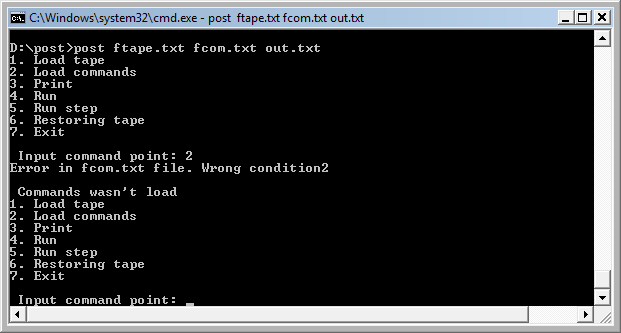
2. ? 1

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



7) Отсутствие команды остановки программы: (нет 5. .)

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

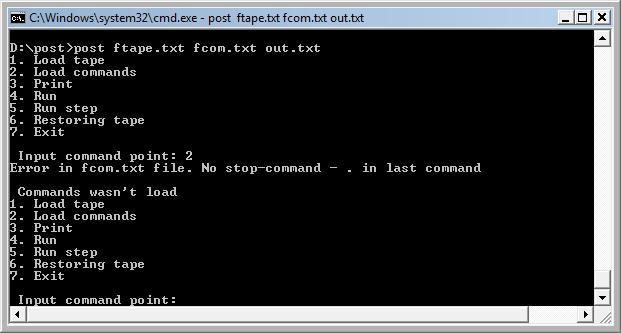
1. < 2

2. ? 1,3

3. > 4

4. 1 5

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



8) Ссылка на несуществующую метку 2. ? 1,30

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

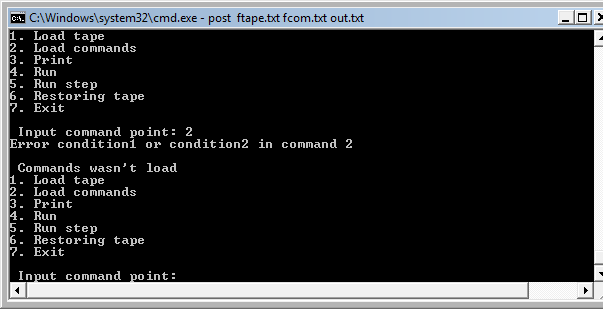
2. ? 1,30

3. > 4

4. 1 5

5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



9) Нормальная загрузка команд

fcom.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 2

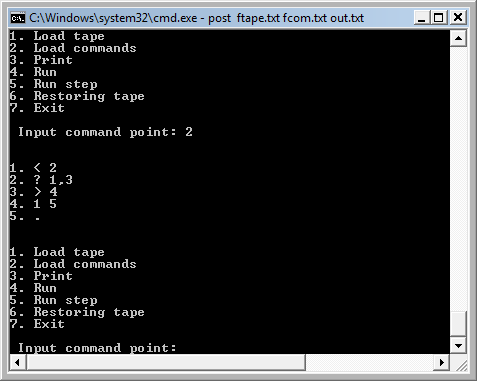
2. ? 1,3

3. > 4

4. 1 5

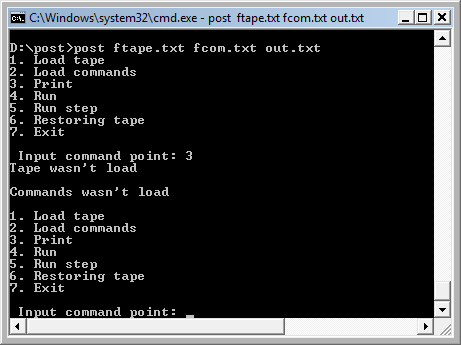
5. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

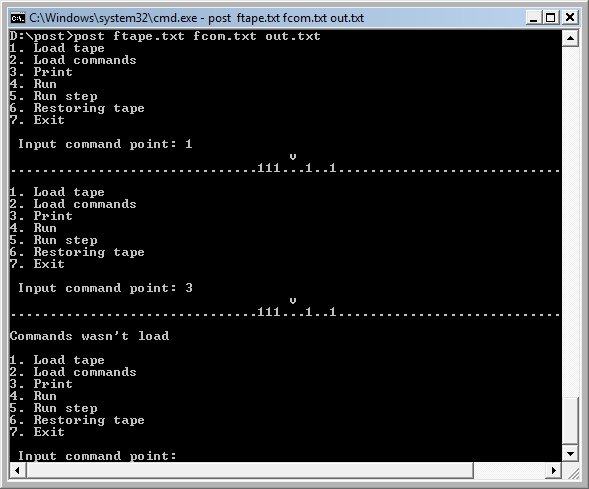


**4.4. Печать**

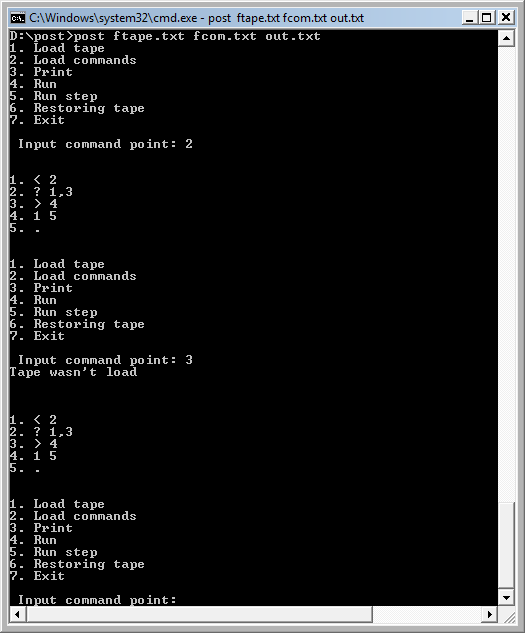
1) Печать без загрузки ленты и команд



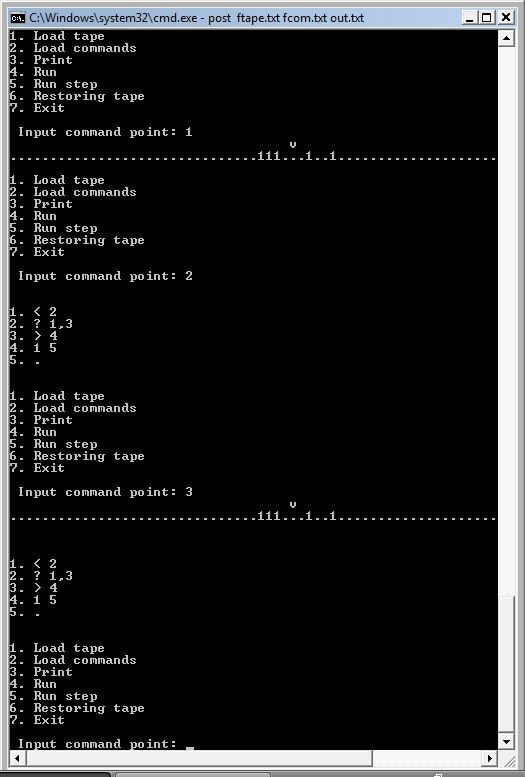
2) Печать без загрузки команд



3) Печать без загрузки ленты

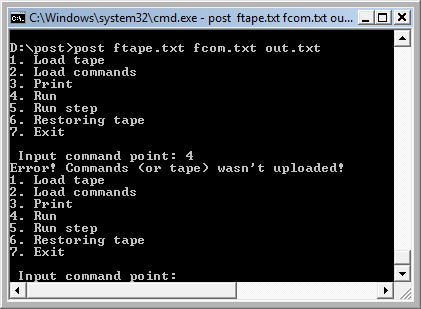


4) Печать загруженных ленты и команд

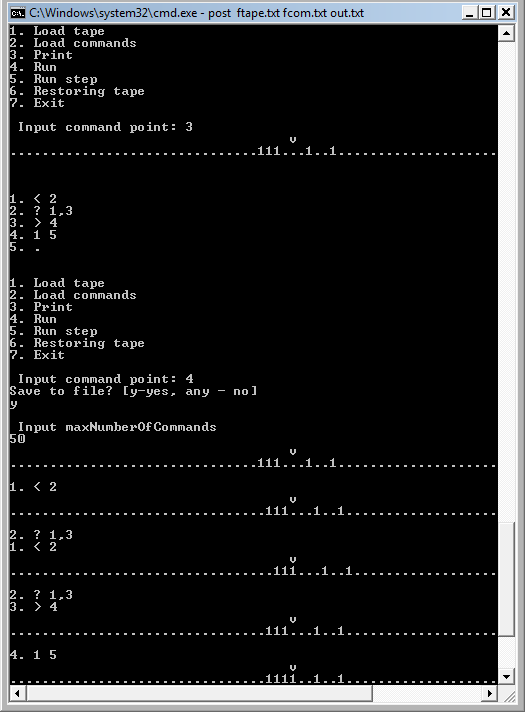


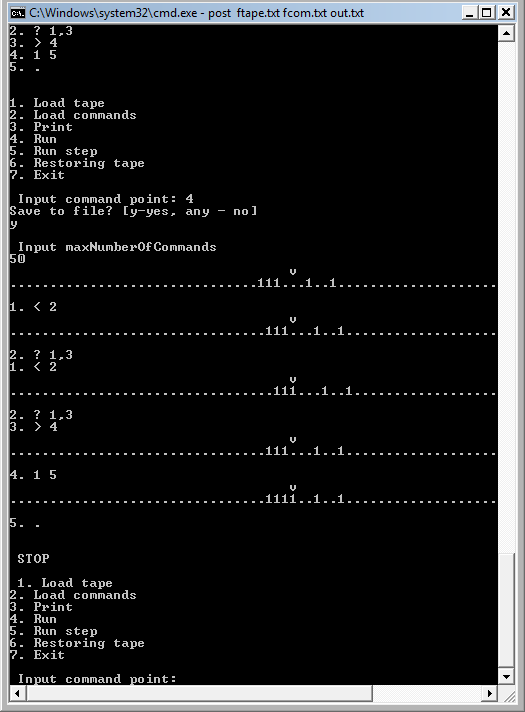
**4.5. Мгновенное выполнение программы**

1) Без загрузки данных (ленты и/или команд)



2) Выполнение с загрузкой всех данных





Файл вывода out.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

                   v

…………………...111…1..1.............................

1. < 2

                    v

................................111...1..1............................

2. ? 1,3

1. < 2

                   v

.................................111...1..1...........................

2. ? 1,3

3. > 4

                   v

................................111...1..1............................

4. 1 5

                   v

................................1111..1..1............................

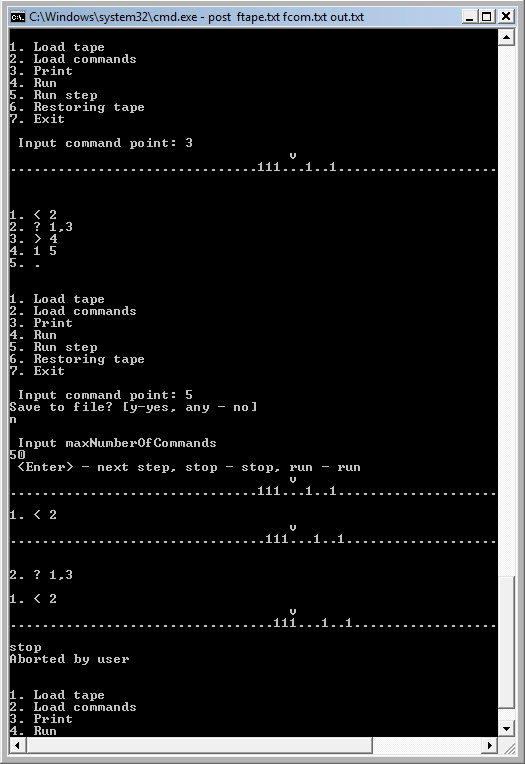
5. .

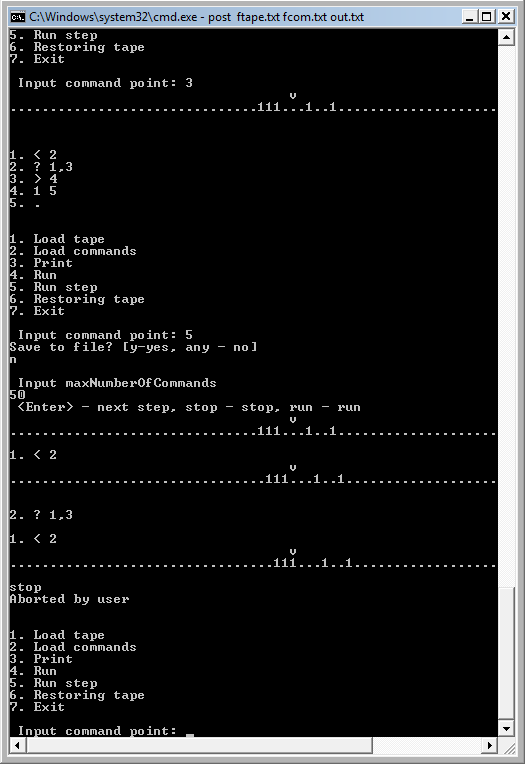
STOP

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

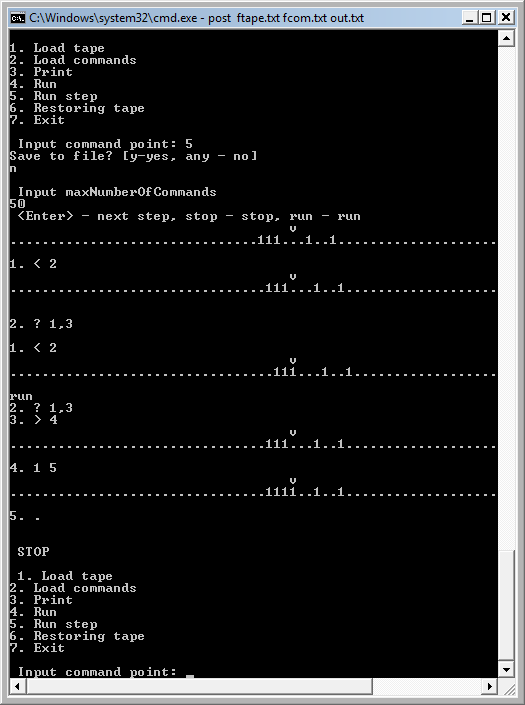
**4.6. Пошаговое выполнение программы**

1) Прерывание отладки





2) Выполнение до конца



3) Защита от зацикливания

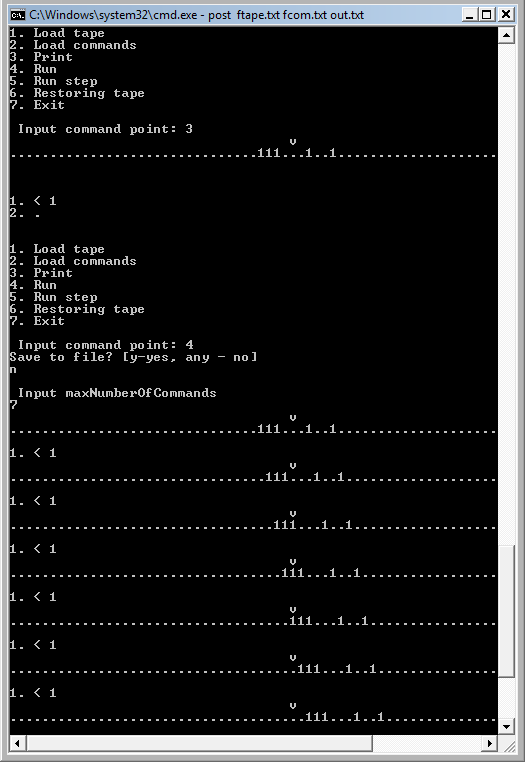
fcom.txt:

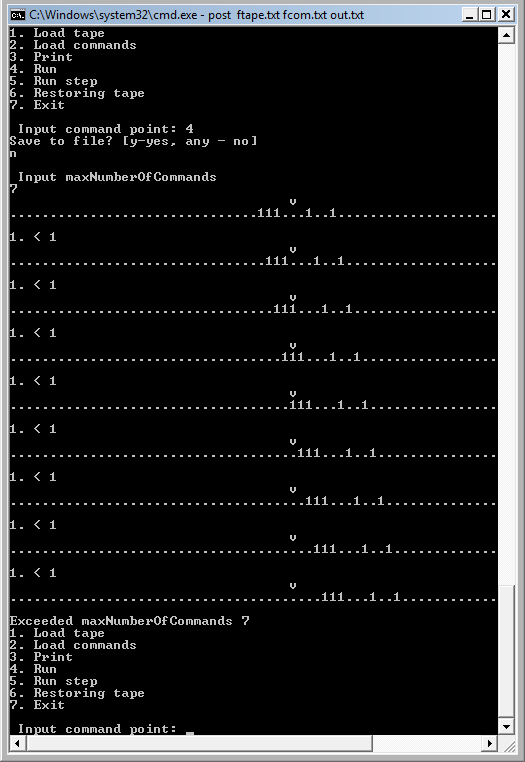
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. < 1

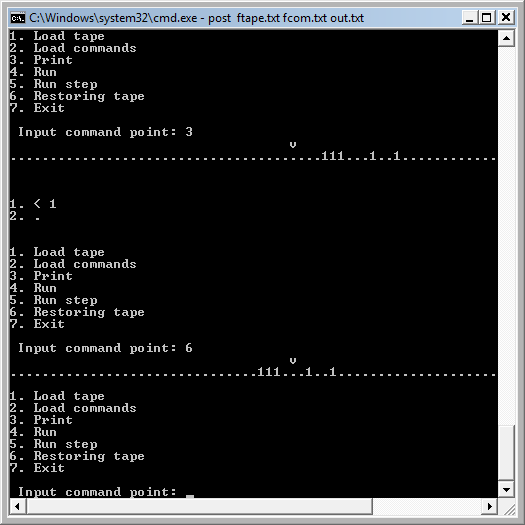
2. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_





**4.7. Восстановление ленты**



**4.8. Нормальная работа программы**

Пример: сложение двух унарных чисел, с записью в файл вывода.

ftape1.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

v

111011

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fcom1.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. ? 3,2

2. > 1

3. 1 4

4. > 5

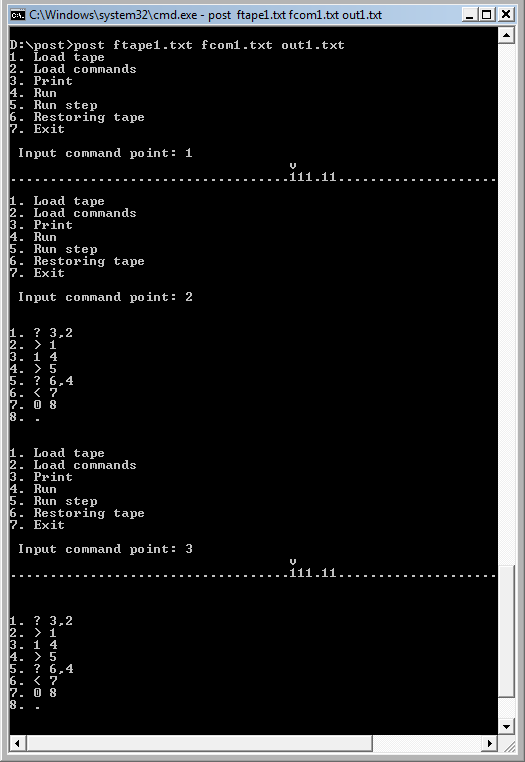
5. ? 6,4

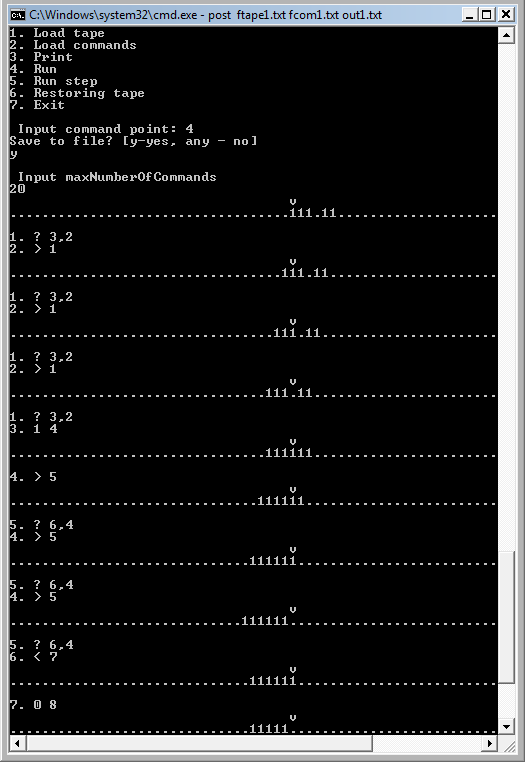
6. < 7

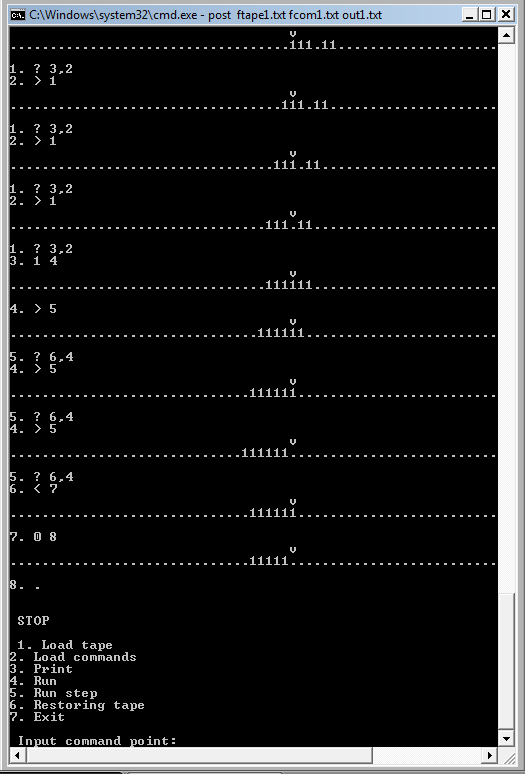
7. 0 8

8. .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_







out1.txt:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

                  v

...................................111.11.............................

1. ? 3,2

2. > 1

                  v

..................................111.11..............................

1. ? 3,2

2. > 1

                  v

.................................111.11...............................

1. ? 3,2

2. > 1

                   v

................................111.11................................

1. ? 3,2

3. 1 4

                   v

................................111111................................

4. > 5

                   v

...............................111111.................................

5. ? 6,4

4. > 5

                    v

..............................111111..................................

5. ? 6,4

4. > 5

                    v

.............................111111...................................

5. ? 6,4

6. < 7

                    v

..............................111111..................................

7. 0 8

                    v

..............................11111...................................

8. .

STOP

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте мною была написана программа, реализующая симулятор машин Поста.

Она поддерживает базовый функционал, необходимый для корректной работы программы, а также полностью соответствует заданным мне в индивидуальном задании требованиям.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: [Вильямс](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2015. — 304 с.
2. Герберт Шилдт. Полный справочник по C, 4-е издание. — Москва: Вильямс, 2004. — 800 с.

# Е. Д. Кожухарь, Многофайловый проект. Автоматизация сборки проекта // ppt-online.org — 2016.

URL: https://ppt-online.org/17179

# Черновик стандарта C11.

# Различные циклы уроков (tutorials) по make (например, http://habrahabr.ru/post/211751).