Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра Вычислительных Систем

Отчет по лабораторной работе

по дисциплине «Архитектура ЭВМ»

Лабораторная работа №3

«Консоль управления моделью Simple Computer. Текстовая часть.»

Выполнил: студент 2 курса группы ИП-811 Мироненко К. А

Проверила: Ткачева Т.А.

Оглавление

[1. Постановка задачи 3](#_Toc42212140)

[2. Описание алгоритма основной программы 5](#_Toc42212141)

[3. Примеры работы программы 6](#_Toc42212142)

[*Приложение* Листинг 8](#_Toc42212143)

# Постановка задачи

*Цель работы*

Изучить принципы работы терминалов ЭВМ в текстовом режиме. Понять, каким образом кодируется текстовая информация и как с помощью неѐ можно управлять работой терминалов. Разработать библиотеку функций myTerm, включающую базовые функции по управлению текстовым терминалом (очистка экрана, позиционирование курсора, управления цветом). Начать разрабатывать консоль управления Simple Computer (вывести на экран текстовую часть).

*Задание на лабораторную работу*

1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграфы 5.4 и 5.5. Изучите страницу man для команды infocmp, базы terminfo, функции ioctl.
2. Откройте текстовый терминал и запустите оболочку bash (оболочка запускается автоматически). Используя команду infocmp, определите (и перепишите их себе) escape-последовательности для терминала, выполняющие следующие действия:
   1. очистка экрана и перемещение курсора в левый верхний угол (clear\_screen);
   2. перемещение курсора в заданную позицию экрана (cursor\_address);
   3. задание цвета последующих выводимых символов (set\_a\_background);
   4. определение цвета фона для последующих выводимых символов (set\_a\_foreground);
   5. скрытие и восстановление курсора (cursor\_invisible, cursor\_visible).
3. Используя оболочку bash, проверьте работу полученных последовательностей. Символ escape задается как \033 или \E. Например – echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
   1. очищает экран;
   2. выводит в пятой строке, начиная с 10 символа Ваше имя красными буквами на черном фоне;
   3. в шестой строке, начиная с 8 символа Вашу группу зеленым цветом на белом фоне;
   4. перемещает курсор в 10 строку, 1 символ и возвращает настройки цвета в значения «по умолчанию».
4. Разработать следующие функции:
   1. int mt\_clrscr (void)- производит очистку и перемещение курсора в левый верхний угол экрана;
   2. int mt\_gotoXY (int, int) - перемещает курсор в указанную позицию. Первый параметр номер строки, второй - номер столбца;
   3. int mt\_getscreensize (int \* rows, int \* cols) - определяет размер экрана терминала (количество строк и столбцов);
   4. int mt\_setfgcolor (enum colors) - устанавливает цвет последующих выводимых символов. В качестве параметра передаѐтся константа из созданного Вами перечислимого типа colors, описывающего цвета терминала;
   5. int mt\_setbgcolor (enum colors) - устанавливает цвет фона последующих выводимых символов. В качестве параметра передаѐтся константа из созданного Вами перечислимого типа colors, описывающего цвета терминала.

*\*Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода.*

1. Оформите разработанные функции как статическую библиотеку myTerm. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

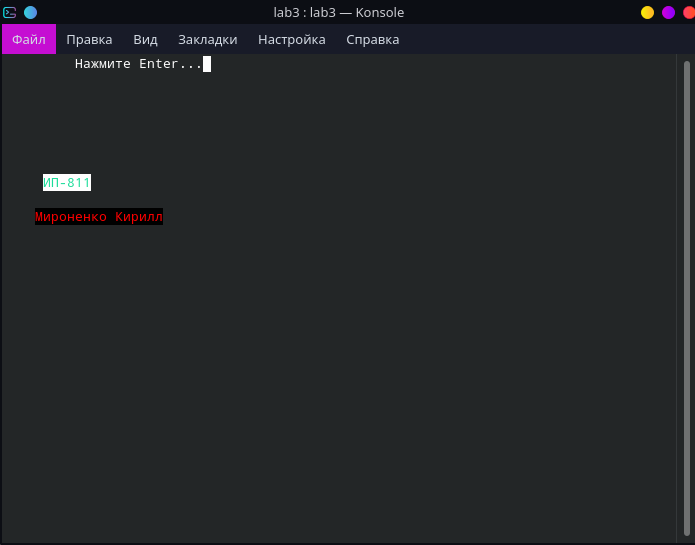
Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях), а также программу, выводящую на экран содержимое оперативной памяти, регистров и назначение клавиш.

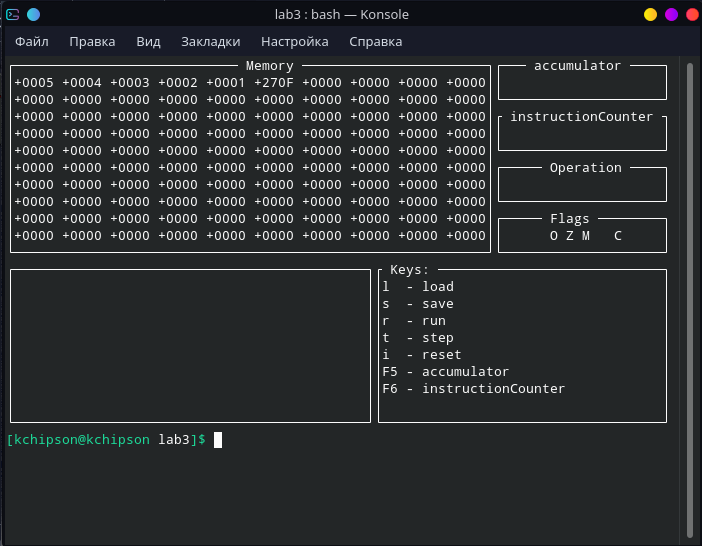
# Описание алгоритма основной программы

* + - * **myTerm**
* Добавлена функция *int mt\_setDefaultColorSettings()*, задающая сбрасывающая настройки терминала до стандартных
* Добавлен дополнительный hpp/cpp файл – myUI, отвечающий за «отрисовку» всей выводимой информации

Остальная организация файлов, их функции и их функциональность соответствуют описанию в методических материалах.

# Примеры работы программы





# *Приложение* Листинг

**main.сpp**

#include "iostream"

#include "SimpleComputer.hpp"

#include "myTerm.hpp"

#include "myBigChars.hpp"

#include "myUI.hpp"

void testRAM() ;

void testCOLOR() ;

int main(){

int value ;

sc\_memoryInit() ;

sc\_regInit() ;

// testRAM() ;

testCOLOR();

return 0 ;

}

void testRAM(){

int value ;

sc\_memorySet(1,5) ;

sc\_memorySet(2,4) ;

sc\_memorySet(3,3) ;

sc\_memorySet(4,2) ;

sc\_memorySet(5,1) ;

for (int i = 0 ; i < 5 ; i++) {

sc\_memoryGet(i + 1, &value) ;

std::cout << "RAM[" << i + 1 << "] = " << value << "\n" ;

}

sc\_memorySave("test.bin") ;

std::cout << "Сохранение файла" << "\n" ;

sc\_memorySet(1,99) ;

sc\_memoryGet(1, &value) ;

std::cout << "RAM[" << 1 << "] = " << value << "\n" ;

sc\_memoryLoad("test.bin") ;

std::cout << "Чтение файла" << "\n" ;

for (int i = 0 ; i < 5 ; i++) {

sc\_memoryGet(i + 1, &value) ;

std::cout << "RAM[" << i + 1 << "] = " << value << "\n" ;

}

std::cout << "\n\n" ;

sc\_regInit() ;

sc\_regSet(OVERFLOW, 1) ;

sc\_regSet(DIVISION\_ERR\_BY\_ZERO, 1) ;

sc\_regSet(OUT\_OF\_MEMORY, 1) ;

sc\_regSet(INCORRECT\_COMMAND, 1) ;

sc\_regGet(OVERFLOW, &value) ;

std::cout << "Флаг \"Переполнение при выполнении операции\": " << value << "\n" ;

sc\_regGet(DIVISION\_ERR\_BY\_ZERO, &value) ;

std::cout << "Флаг \"Ошибка деления на 0\": " << value << "\n" ;

sc\_regGet(OUT\_OF\_MEMORY, &value) ;

std::cout << "Флаг \"Ошибка выхода за границы памяти\": " << value << "\n" ;

sc\_regGet(IGNORING\_TACT\_PULSES, &value) ;

std::cout << "Флаг \"Игнорирование тактовых импульсов\": " << value << "\n" ;

sc\_regGet(INCORRECT\_COMMAND, &value) ;

std::cout << "Флаг \"Указана неверная команда\": " << value << "\n" ;

std::cout << "\n\n" ;

int f = 0, f\_c = 0, f\_o = 0 ;

sc\_commandEncode(0x33, 0x59, &f) ;

std::cout << "Закодированная команда: "<< f << "\n" ;

sc\_commandDecode(f, &f\_c, &f\_o) ;

std::cout << "Команда: "<< std::hex << f\_c << " операнд: "<< std::hex << f\_o << "\n" ;

}

void testCOLOR(){

mt\_clrScreen() ;

mt\_gotoXY(5, 10) ;

mt\_setBGcolor(BLACK) ;

mt\_setFGcolor(RED) ;

printf("Мироненко Кирилл") ;

mt\_gotoXY(6, 8) ;

mt\_setBGcolor(WHITE) ;

mt\_setFGcolor(GREEN) ;

printf("ИП-811") ;

mt\_gotoXY(10, 1) ;

mt\_setDefaultColorSettings() ;

printf("Нажмите Enter...") ;

getchar() ;

ui\_initial() ;

sc\_memorySet(0,5) ;

sc\_memorySet(1,4) ;

sc\_memorySet(2,3) ;

sc\_memorySet(3,2) ;

sc\_memorySet(4,1) ;

sc\_memorySet(5,9999) ;

sc\_regSet(OVERFLOW, 1) ;

sc\_regSet(DIVISION\_ERR\_BY\_ZERO, 1) ;

sc\_regSet(OUT\_OF\_MEMORY, 1) ;

sc\_regSet(INCORRECT\_COMMAND, 1) ;

ui\_update() ;

}

**SimpleComputer.hpp**

#ifndef SIMPLECOMPUTER\_HPP

#define SIMPLECOMPUTER\_HPP

#include "iostream"

#include <fstream>

#include <stdexcept>

#define OVERFLOW 0 // Переполнение при выполнении операции

#define DIVISION\_ERR\_BY\_ZERO 1 // Ошибка деления на 0

#define OUT\_OF\_MEMORY 2 // Ошибка выхода за границы памяти

#define IGNORING\_TACT\_PULSES 3 // Игнорирование тактовых импульсов

#define INCORRECT\_COMMAND 4 // Указана неверная команда

const int SC\_REG\_SIZE = 5 ;

const int SC\_MEM\_SIZE = 100 ;

extern int sc\_memory[SC\_MEM\_SIZE] ;

extern uint8\_t sc\_regFLAGS ;

int sc\_memoryInit () ;

int sc\_memorySet (int address, int value) ;

int sc\_memoryGet (int address, int \* value) ;

int sc\_memorySave (const std::string& filename) ;

int sc\_memoryLoad (const std::string& filename) ;

int sc\_regInit () ;

int sc\_regSet (int reg, int value) ;

int sc\_regGet (int reg, int \* value) ;

int sc\_commandEncode (int command, int operand, int \* value) ;

int sc\_commandDecode (int value, int \* command, int \* operand) ;

#endif //SIMPLECOMPUTER\_HPP

**SimpleComputer.cpp**

#include "SimpleComputer.hpp"

int sc\_memory[SC\_MEM\_SIZE] ;

uint8\_t sc\_regFLAGS ;

/// Инициализирует оперативную память SC, задавая всем её ячейкам нулевые значения

/// \return 0

int sc\_memoryInit()

{

for (int i = 0 ; i < SC\_MEM\_SIZE ; i++)

sc\_memory[i] = 0 ;

return 0 ;

}

/// Задает значение указанной ячейки памяти

/// \param address - ячейка памяти

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_memorySet(int address, int value){

try

{

if (address < 0 || address >= SC\_MEM\_SIZE)

throw std::overflow\_error("ERROR\tАдрес выходит за границу памяти!") ;

sc\_memory[address] = value ;

return 0 ;

}

catch (std::overflow\_error err)

{

sc\_regSet(OUT\_OF\_MEMORY, 1) ;

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Возвращает значение указанной ячейки памяти

/// \param address - ячейка памяти

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_memoryGet(int address, int \* value){

try {

if (address < 0 || address >= SC\_MEM\_SIZE){

throw std::overflow\_error("ERROR\tАдрес выходит за границу памяти") ;

}

\*value = sc\_memory[address] ;

return 0 ;

}

catch (std::overflow\_error err)

{

sc\_regSet(OUT\_OF\_MEMORY, 1) ;

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Сохраняет содержимое памяти в файл в бинарном виде

/// \param filename - имя файла

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_memorySave(const std::string& filename){

std::ofstream out(filename, std::ios::binary|std::ios::out) ;

try{

if (!out.is\_open()) {

throw std::runtime\_error("ERROR\tНе удалось открыть файл \'" + filename + "\'") ;

}

out.write((char\*)&(sc\_memory), sizeof(sc\_memory)) ;

out.close() ;

return 0 ;

}

catch (std::runtime\_error err) {

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Загружает из указанного файла содержимое оперативной памяти

/// \param filename - имя файла

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_memoryLoad(const std::string& filename){

std::ifstream in(filename, std::ios::binary|std::ios::out) ;

try{

if (!in.is\_open()) {

throw std::runtime\_error("ERROR\tНе удалось открыть файл \'" + filename + "\'") ;

}

in.read((char\*)&(sc\_memory), sizeof(sc\_memory)) ;

in.close() ;

return 0 ;

}

catch (std::runtime\_error err) {

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Инициализирует регистр флагов нулевым значением

/// \return 0

int sc\_regInit(){

sc\_regFLAGS = 0 ;

return 0 ;

}

/// Устанавливает значение указанного регистра флагов

/// \param reg - флаг

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_regSet(int reg, int value){

try{

if (reg < 0 || reg >= SC\_REG\_SIZE)

throw std::overflow\_error("ERROR\tНедопустимый регистр") ;

if (value != 0 && value != 1)

throw std::invalid\_argument("ERROR\tНекорректное значение, допустимы: 0, 1") ;

value == 1 ? (sc\_regFLAGS |= (1 << reg)) : (sc\_regFLAGS &= ~(1 << reg)) ;

return 0 ;

}

catch (std::exception err)

{

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Возвращает значение указанного флага

/// \param reg - флаг

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_regGet(int reg, int \*value){

try {

if (reg < 0 || reg >= SC\_REG\_SIZE)

throw std::overflow\_error("Недопустимый регистр") ;

(sc\_regFLAGS & (1 << reg)) ? \*value = 1 : \*value = 0 ;

return 0 ;

}

catch (std::overflow\_error err)

{

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Кодирует команду с указанным номером и операндом и помещает результат в value

/// \param command - команда

/// \param operand - операнд

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_commandEncode(int command, int operand, int \* value){

try {

if (!(command > 0x9 && command < 0x12) && !(command > 0x19 && command < 0x22) && !(command > 0x29 && command < 0x34) && !(command > 0x39 && command < 0x77))

throw std::invalid\_argument("ERROR\tНедопустимая команда") ;

if ((operand < 0) || (operand >= 128))

throw std::invalid\_argument("ERROR\tНедопустимый операнд") ;

\* value = 0 ;

/\* Операнд \*/

for (int i = 0 ; i < 7 ; i++) {

int bit = (operand >> i) & 1 ;

\*value |= (bit << i) ;

}

/\* Команда \*/

for (int i = 0 ; i < 7 ; i++) {

int8\_t bit = (command >> i) & 1 ;

\*value |= (bit << (i + 7)) ;

}

return 0 ;

}

catch (std::invalid\_argument err){

std::cout << err.what() << std::endl ;

return -1 ;

}

}

/// Декодирует значение как команду SС

/// \param value - значение

/// \param command - команда

/// \param operand - операнд

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int sc\_commandDecode(int value, int \* command, int \* operand){

try {

int tmpCom = 0, tmpOp = 0 ;

if ((value >> 14) & 1)

throw std::invalid\_argument("ERROR\tНе является командой") ;

for (int i = 0 ; i < 7 ; i++) {

int bit = (value >> i) & 1 ;

tmpOp |= (bit << i) ;

}

for (int i = 0 ; i < 7 ; i++) {

int bit = (value >> (i + 7)) & 1 ;

tmpCom |= (bit << i) ;

}

if (!(tmpCom > 0x9 && tmpCom < 0x12) && !(tmpCom > 0x19 && tmpCom < 0x22) && !(tmpCom > 0x29 && tmpCom < 0x34) && !(tmpCom > 0x39 && tmpCom < 0x77))

throw std::invalid\_argument("ERROR\tНе удалось декодировать команду") ;

if ((tmpOp < 0) || (tmpOp >= 128))

throw std::invalid\_argument("ERROR\tНе удалось декодировать операнд") ;

\* command = tmpCom ;

\* operand = tmpOp ;

return 0 ;

}

catch (std::invalid\_argument err){

sc\_regSet(INCORRECT\_COMMAND, 1) ;

std::cout << err.what() << std::endl ;

return -1 ;

}

}

**myBigChars.hpp**

#ifndef MYBIGCHARS\_HPP

#define MYBIGCHARS\_HPP

#include "myTerm.hpp"

/\* Псевдографика \*/

#define ACS\_CKBOARD "a" // Штриховка

#define ACS\_ULCORNER "l" // Левый верхний угол

#define ACS\_URCORNER "k" // Правый верхний угол

#define ACS\_LRCORNER "j" // Правый нижний угол

#define ACS\_LLCORNER "m" // Левый нижний угол

#define ACS\_HLINE "q" // Горизонтальная линия

#define ACS\_VLINE "x" // Вертикальная линия

int bc\_printA (char \* str) ;

int bc\_box(int x, int y, int width, int height) ;

int bc\_printBigChar(int \*big, int x, int y, enum colors colorFG, enum colors colorBG) ;

int bc\_setBigCharPos (int \* big, int x, int y, int value) ;

int bc\_getbigcharpos(int \* big, int x, int y, int \*value) ;

int bc\_bigCharWrite(int fd, int \* big, int count) ;

int bc\_bigCharRead(int fd, int \* big, int need\_count, int \* count) ;

#endif //MYBIGCHARS\_HPP

**myBigChars.cpp**

#include "myBigChars.hpp"

/// Выводит строку символов с использованием дополнительной кодировочной таблицы

/// \param str - символ

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_printA (char \* str){

printf("\033(0%s\033(B", str) ;

return 0 ;

}

/// Выводит на экран псевдографическую рамку

/// \param x - строка левого вернего угла рамки

/// \param y - столбец левого вернего угла рамки

/// \param width - ширина рамки

/// \param height - высота рамки

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_box(int x, int y, int width, int height){

int rows, cols ;

mt\_getScreenSize(&rows, &cols) ;

if ((x <= 0) || (y <= 0) || (x + width - 1 > cols) || (y + height - 1 > rows) || (width <= 1) || (height <= 1))

return -1 ;

mt\_gotoXY(x, y) ;

bc\_printA((char\*)ACS\_ULCORNER) ;

mt\_gotoXY(x + width - 1, y);

bc\_printA((char\*)ACS\_URCORNER);

mt\_gotoXY(x + width - 1, y + height - 1) ;

bc\_printA((char\*)ACS\_LRCORNER) ;

mt\_gotoXY(x, y + height - 1) ;

bc\_printA((char\*)ACS\_LLCORNER) ;

/\* Горизонтальные линии \*/

for (int i = 1; i < width - 1; ++i) {

// верхняя

mt\_gotoXY(x + i, y) ;

bc\_printA((char\*)ACS\_HLINE) ;

// нижняя

mt\_gotoXY(x + i, y + height - 1) ;

bc\_printA((char\*)ACS\_HLINE) ;

}

/\* Вертикальные линии \*/

for (int i = 1; i < height - 1; ++i) {

// верхняя

mt\_gotoXY(x, y + i) ;

bc\_printA((char\*)ACS\_VLINE) ;

// нижняя

mt\_gotoXY(x + width - 1, y + i) ;

bc\_printA((char\*)ACS\_VLINE) ;

}

return 0 ;

}

/// Выводит на экран "большой символ" размером восемь строк на восемь столбцов

/// \param big

/// \param x - строка левого вернего угла символа

/// \param y - столбец левого вернего угла символа

/// \param colorFG - цвет текста

/// \param colorBG - цвет фона

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_printBigChar(int \*big, int x, int y, enum colors colorFG, enum colors colorBG){

return 0 ;

}

/// Устанавливает значение знакоместа "большого символа"

/// \param big

/// \param x - строка

/// \param y - столбец

/// \param value - значение

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_setBigCharPos (int \* big, int x, int y, int value){

return 0 ;

}

/// Возвращает значение позиции в "большом символе"

/// \param big

/// \param x - строка

/// \param y - столбец

/// \param value

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_getbigcharpos(int \* big, int x, int y, int \*value){

return 0 ;

}

/// Записывает заданное число "больших символов" в файл. Формат записи определяется пользователем;

/// \param fd

/// \param big

/// \param count

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_bigCharWrite(int fd, int \* big, int count){

return 0 ;

}

/// Cчитывает из файла заданное количество "больших символов"

/// Третий параметр указывает адрес переменной, в которую помещается количество считанных символов или 0, в случае ошибки.

/// \param fd

/// \param big

/// \param need\_count

/// \param count

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int bc\_bigCharRead(int fd, int \* big, int need\_count, int \* count){

return 0 ;

}

**myTerm.hpp**

#ifndef MYTERM\_HPP

#define MYTERM\_HPP

#include <iostream>

#include <sys/ioctl.h>

enum colors {RED = 196, GREEN = 10, BLUE = 20, BLACK = 16, WHITE = 15};

int mt\_clrScreen () ;

int mt\_gotoXY(int col, int row) ;

int mt\_getScreenSize(int\* rows, int\* cols) ;

int mt\_setFGcolor(enum colors color) ;

int mt\_setBGcolor(enum colors color) ;

int mt\_setDefaultColorSettings() ;

#endif //MYTERM\_HPP

**myTerm.cpp**

#include "myTerm.hpp"

/// Производит очистку и перемещение курсора в левый верхний угол экрана

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_clrScreen (){

printf("\033[H\033[2J") ;

return 0 ;

}

/// Перемещает курсор в указанную позицию

/// \param col - столбец

/// \param row - строка

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_gotoXY(int col, int row)

{

int rows, cols ;

if (mt\_getScreenSize(&rows, &cols) == -1)

return -1 ;

if ((row > rows) || (row <= 0)||(col > cols) || (col <= 0))

return -1 ;

printf("\033[%d;%dH", row, col) ;

return 0 ;

}

/// Определяет размер экрана терминала

/// \param rows - кол-во строк

/// \param cols - кол-во столбцов

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_getScreenSize(int\* rows, int\* cols){

struct winsize ws ;

if (ioctl(1, TIOCGWINSZ, &ws))

return -1 ;

\* rows = ws.ws\_row ;

\* cols = ws.ws\_col ;

return 0 ;

}

/// Устанавливает цвет последующих выводимых символов

/// \param color - цвет из перечисления colors

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_setFGcolor(enum colors color){

printf("\033[38;5;%dm",color) ;

return 0 ;

}

/// Устанавливает цвет фона последующих выводимых символов

/// \param color - цвет из перечисления colors

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_setBGcolor(enum colors color){

printf("\033[48;5;%dm",color) ;

return 0 ;

}

/// Возвращает цвета в стандартное состояние

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int mt\_setDefaultColorSettings(){

printf("\033[0m") ;

return 0 ;

}

**myUI.hpp**

#ifndef MYUI\_HPP

#define MYUI\_HPP

#include "myBigChars.hpp"

#include "SimpleComputer.hpp"

int ui\_initial() ;

int ui\_update() ;

**myUI.cpp**

#include "myUI.hpp"

/// Отрисовка "боксов"

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int drawingBoxes(){

if (bc\_box(1, 1, 61, 12)) // Окно Memory

return -1 ;

if (bc\_box(62, 1, 22, 3)) // Окно accumulator

return -1 ;

if (bc\_box(62, 4, 22, 3)) // Окно instructionCounter

return -1 ;

if (bc\_box(62, 7, 22, 3)) // Окно Operation

return -1 ;

if (bc\_box(62, 10, 22, 3)) // Окно Flags

return -1 ;

if (bc\_box(1, 13, 46, 10)) // Окно BigChars

return -1 ;

if (bc\_box(47, 13, 37, 10)) // Окно Keys

return -1 ;

return 0 ;

}

/// Отрисовка заголовков и текста

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int drawingTexts(){

/\* Заголовки \*/

mt\_gotoXY(30,1) ;

printf(" Memory ") ;

mt\_gotoXY(66,1) ;

printf(" accumulator ") ;

mt\_gotoXY(63,4) ;

printf(" instructionCounter ") ;

mt\_gotoXY(68,7) ;

printf(" Operation ") ;

mt\_gotoXY(68,10) ;

printf(" Flags ") ;

mt\_gotoXY(48,13) ;

printf(" Keys: ") ;

/\* HotKeys \*/

char\* hotK[] = {(char \*)"l - load",

(char \*)"s - save",

(char \*)"r - run",

(char \*)"t - step",

(char \*)"i - reset",

(char \*)"F5 - accumulator",

(char \*)"F6 - instructionCounter"};

for (int i = 0 ; i < sizeof(hotK) / sizeof(\*hotK) ; ++i) {

mt\_gotoXY(48,i + 14) ;

printf("%s", hotK[i]) ;

}

return 0 ;

}

/// Отрисовка памяти

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int drawingMemory(){

for (int i = 0 ; i < 10 ; ++i)

for (int j = 0 ; j < 10 ; ++j) {

mt\_gotoXY(2 + (5 \* j + j), 2 + i) ;

int tmp = sc\_memory[i \* 10 + j] ;

if((tmp >> 14) & 1)

printf(" %04X", tmp & (~ (1 << 14))) ;

else

printf("+%04X", tmp) ;

}

return 0 ;

}

/// Отрисовка флагов

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int drawingFlags(){

char tmp[] = {'O', 'Z', 'M', 'P', 'C'};

for (int i = 0 ; i < SC\_REG\_SIZE ; ++i) {

int value ;

if (sc\_regGet(i, &value))

return -1 ;

if (value){

mt\_gotoXY(69 + (i \* 2), 11) ;

printf("%c", tmp[i]) ;

}

}

return 0 ;

}

/// Начальная отрисовка интерфейса пользователя

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int ui\_initial(){

mt\_clrScreen() ;

if (drawingBoxes())

return -1 ;

if (drawingTexts())

return -1 ;

mt\_gotoXY(1, 23) ;

return 0 ;

}

/// Обновление интерфейса пользователя

/// \return 0 - в случае успешного выполнения, -1 - в случае ошибки

int ui\_update(){

if (drawingMemory())

return -1 ;

if (drawingFlags())

return -1 ;

mt\_gotoXY(1, 23) ;

return 0 ;

}