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 №5

«Консоль управления моделью Simple Computer. Клавиатура. Обработка нажатия клавиш. Неканонический режим работы терминала »

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

Проверила: Ткачева Т.А.

**Оглавление**

[1. Постановка задачи 3](#_Toc42253906)

[2. Описание алгоритма основной программы 5](#_Toc42253907)

[3. Примеры работы программы 6](#_Toc42253908)

[*Приложение* Листинг 9](#_Toc42253909)

# Постановка задачи

*Цель работы*

Изучить устройство клавиатуры и принципы обработки нажатия клавиш в текстовом терминале. Создать «распознаватель» нажатой клавиши по формируемой последовательности символов. Разработать библиотеку myReadkey. Доработать интерфейс консоли управления Simple Computer так, чтобы можно было изменять значения ячеек памяти и регистров.

*Задание на лабораторную работу*

1. Прочитайте главу 5 практикума по курсу «Организация ЭВМ и систем». Обратите особое внимание на параграф 5.1. Изучите страницу man для команд infocmp и read, базы terminfo.
2. Используя оболочку bash и команду read, определите последовательности, формируемые нажатием на буквенно-цифровые, функциональные клавиши и клавиши управления курсором. Используя команду infocmp, убедитесь, что получены правильные последовательности символов, генерируемые функциональными клавишами «F5» и «F6».
3. Разработайте функции:

* int rk\_readkey (enum keys \*) - анализирующую последовательность символов (возвращаемых функцией read при чтении с терминала) и возвращающую первую клавишу, которую нажал пользователь. В качестве параметра в функцию передаётся адрес переменной, в которую возвращается номер нажатой (enum keys – перечисление распознаваемых клавиш);
* int rk\_mytermsave (void) - сохраняет текущие параметры терминала;
* int rk\_mytermrestore (void) - восстанавливает сохранѐнные параметры терминала.
* int rk\_mytermregime (int regime, int vtime, int vmin, int echo, int sigint) - переключает терминала между режимами. Для неканонического режима используются значения второго и последующего параметров.

*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 myReadkey. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

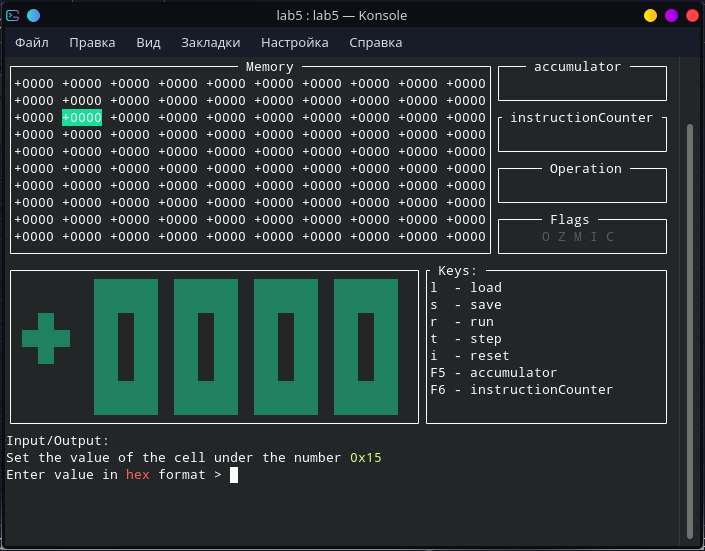
Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях). Необходимо доработать программу лабораторной работы 3, выводящую на экран согласно рисунку 1 консоль управления Simple Computer так, чтобы возможно было задавать значения ячейкам оперативной памяти, регистрам и обрабатывалось нажатие клавиш “s”, “l”.

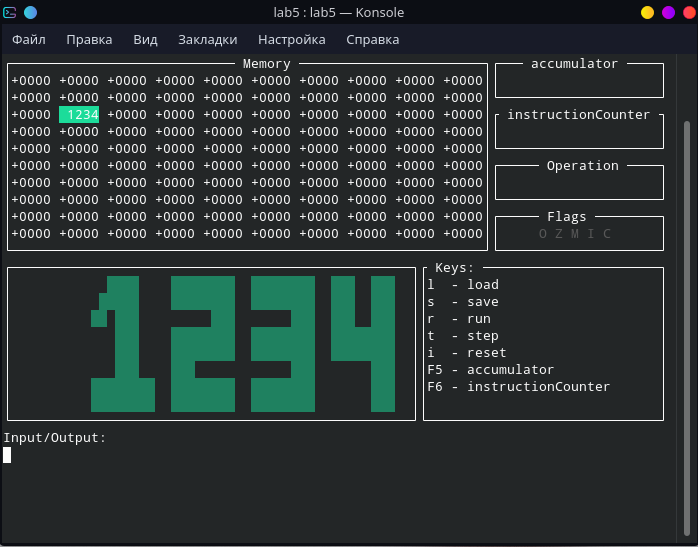
# Описание алгоритма основной программы

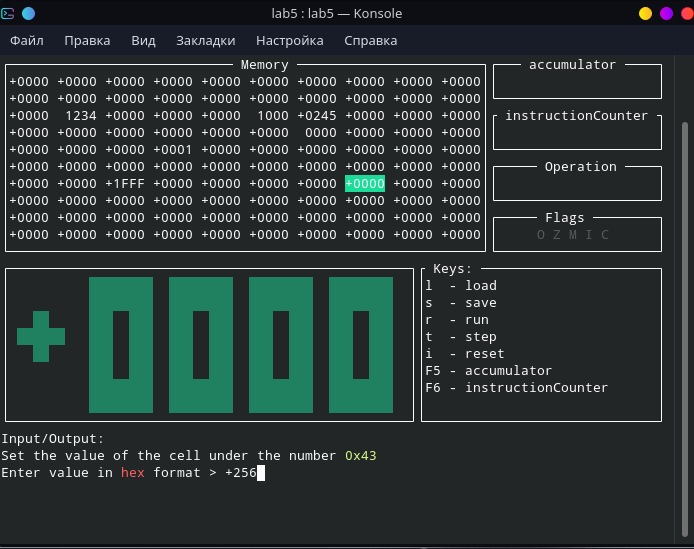
* + - * **myTerm**
* Добавлена функция *int mt\_setDefaultColorSettings()*, сбрасывающая настройки терминала до стандартных;
* Добавлен дополнительный hpp/cpp файл – «myUI», отвечающий за «отрисовку» всей выводимой информации;
* **myUI**
  + Добавлена функция *int ui\_messageOutput(char \*str, enum colors color)*, выводящая сообщение заданным цветом;
* Отрисовка «Big char’а» происходит в «myUI»;
* «Большие символы» статически заданы массивом *unsigned int bc[][2]*
* Регистр «instructionCounter» заменен на «currMemCell», т.к. «instructionCounter» будет отвечать за следующую ячейку на которую перейдет «currMemCell» (исходя из задания курсовой);
* В «main» do-while, проверяющий нажатия клавиш

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

# Примеры работы программы







# *Приложение* Листинг

**main.сpp**

#include <iostream>

#include <fcntl.h>

#include "SimpleComputer.hpp"

#include "myTerm.hpp"

#include "myBigChars.hpp"

#include "myUI.hpp"

#include "myReadkey.hpp"

void testRAM() ;

void testCOLOR() ;

void testBIGCHARS() ;

int main(){

// fprintf (stderr, "Ошибка открытия терминала.\n");

rk\_myTermSave() ;

sc\_memoryInit() ;

sc\_regInit() ;

keys key;

do {

ui\_update() ;

rk\_readKey(&key);

switch(key){

case KEY\_UP: (currMemCell <= 9) ? (currMemCell = 90 + currMemCell) : (currMemCell -= 10) ; break ;

case KEY\_RIGHT: (!((currMemCell + 1) % 10)) ? (currMemCell -= 9) : (currMemCell += 1) ; break ;

case KEY\_DOWN: (currMemCell >= 90) ? (currMemCell = currMemCell - 90) : (currMemCell += 10) ; break ;

case KEY\_LEFT: (!(currMemCell % 10)) ? (currMemCell += 9) : (currMemCell -= 1) ; break ;

case KEY\_L: ui\_loadMemory() ; break ;

case KEY\_S: ui\_saveMemory() ; break ;

case KEY\_R: break ;

case KEY\_T: break ;

case KEY\_I: break ;

case KEY\_F5: break ;

case KEY\_F6: break ;

case KEY\_ENTER: ui\_setMCellValue() ; break ;

case KEY\_OTHER: break;

case KEY\_ESC: break;

}

} while(key != KEY\_ESC) ;

// testRAM() ;

// testCOLOR() ;

// testBIGCHARS() ;

return 0 ;

}

void testRAM(){

short 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) ;

bool reg\_value;

sc\_regGet(OVERFLOW, &reg\_value) ;

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

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

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

sc\_regGet(OUT\_OF\_MEMORY, &reg\_value) ;

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

sc\_regGet(IGNORING\_TACT\_PULSES, &reg\_value) ;

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

sc\_regGet(INCORRECT\_COMMAND, &reg\_value) ;

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

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

short 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\_update() ;

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() ;

}

void testBIGCHARS(){

ui\_update() ;

sc\_memorySet(0,0x9999) ;

sc\_memorySet(11,0x1234) ;

sc\_memorySet(22,0xABCD) ;

ui\_update() ;

printf("Нажмите Enter...") ;

getchar() ;

currMemCell = 11 ;

ui\_update() ;

printf("Нажмите Enter...") ;

getchar() ;

currMemCell = 22 ;

ui\_update() ;

int fd = open("test.txt", O\_WRONLY) ;

bc\_bigCharWrite(fd, \*bc, 18) ;

close(fd) ;

}

**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 short int SC\_REG\_SIZE = 5 ;

const short int SC\_MEM\_SIZE = 100 ;

extern int8\_t currMemCell ;

int sc\_memoryInit () ;

int sc\_memorySet (int8\_t address, short int value) ;

int sc\_memoryGet (int8\_t address, short int \* value) ;

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

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

int sc\_regInit () ;

int sc\_regSet (int8\_t reg, bool value) ;

int sc\_regGet (int8\_t reg, bool \* value) ;

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

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

#endif //SIMPLECOMPUTER\_HPP

**SimpleComputer.сpp**

#include "SimpleComputer.hpp"

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

int8\_t currMemCell ;

uint8\_t sc\_regFLAGS ;

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

/// \return 0

int sc\_memoryInit()

{

currMemCell = 0 ;

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(int8\_t address, short 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(int8\_t address, short 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(int8\_t reg, bool 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(int8\_t reg, bool \*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(short int command, short int operand, short 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(short int value, short int \* command, short 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 ;

}

}

**myTerm.hpp**

#ifndef MYTERM\_HPP

#define MYTERM\_HPP

#include <iostream>

#include <sys/ioctl.h>

enum colors {

RED = 196,

PEACH = 203,

GREEN = 10,

SOFT\_GREEN =192,

BLUE = 20,

BLACK = 16,

GRAY = 240,

WHITE = 15,

DEFAULT = 0};

int mt\_clrScreen () ;

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

int mt\_getScreenSize(unsigned int \*rows, unsigned 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(unsigned int col, unsigned int row)

{

unsigned 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(unsigned int \*rows, unsigned 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 ;

}

**myBigChars.hpp**

#ifndef MYBIGCHARS\_HPP

#define MYBIGCHARS\_HPP

#include <unistd.h>

#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

extern unsigned int bc[][2] ;

int bc\_printA (char ch) ;

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

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

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

int bc\_getbigCharPos(unsigned int \* big, int x, int y, bool \*value) ;

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

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

#endif //MYBIGCHARS\_HPP

**myBigChars.cpp**

#include "myBigChars.hpp"

unsigned int bc[][2] = {{0xE7E7FFFF, 0xFFFFE7E7}, // 0 | 11100111111001111111111111111111 11111111111111111110011111100111

{0x1CDC7C3C, 0xFFFF1C1C}, // 1 | 00011100110111000111110000111100 11111111111111110001110000011100

{0xFF07FFFF, 0xFFFFE0FF}, // 2 | 11111111000001111111111111111111 11111111111111111110000011111111

{0xFF07FFFF, 0xFFFF07FF}, // 3 | 11111111000001111111111111111111 11111111111111110000011111111111

{0xFFE7E7E7, 0x070707FF}, // 4 | 11111111111001111110011111100111 00000111000001110000011111111111

{0xFFE0FFFF, 0xFFFF07FF}, // 5 | 11111111111000001111111111111111 11111111111111110000011111111111

{0xFFE0FFFF, 0xFFFFE7FF}, // 6 | 11111111111000001111111111111111 11111111111111111110011111111111

{0x1C0EFFFE, 0x3838FE38}, // 7 | 00011100000011101111111111111110 00111000001110001111111000111000

{0x7EE7FF7E, 0x7EFFE77E}, // 8 | 01111110111001111111111101111110 01111110111111111110011101111110

{0xFFE7FFFF, 0xFFFF07FF}, // 9 | 11111111111001111111111111111111 11111111111111110000011111111111

{0xFFE7FF7E, 0xE7E7E7FF}, // A | 11111111111001111111111101111110 11100111111001111110011111111111

{0xFEE7FFFE, 0xFEFFE7FE}, // B | 11111110111001111111111111111110 11111110111111111110011111111110

{0xE0E7FF7E, 0x7EFFE7E0}, // C | 11100000111001111111111101111110 01111110111111111110011111100000

{0xE7E7FFF8, 0xF8FFE7E7}, // D | 11100111111001111111111111111000 11111000111111111110011111100111

{0xFFE0FFFF, 0xFFFFE0FF}, // E | 11111111111000001111111111111111 11111111111111111110000011111111

{0xFFE0FFFF, 0xE0E0E0FF}, // F | 11111111111000001111111111111111 11100000111000001110000011111111

{0x7E180000, 0x00000018}, // + | 01111110000110000000000000000000 00000000000000000001100001111110

{0x7E000000, 0x00000000}, // - | 01111110000000000000000000000000 00000000000000000000000000000000

} ;

/// 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 ch){

printf("\033(0%c\033(B", ch) ;

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){

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

if (colorFG != DEFAULT)

mt\_setFGcolor(colorFG) ;

if (colorBG != DEFAULT)

mt\_setBGcolor(colorBG) ;

for (int i = 0; i < 8; ++i)

for (int j = 0; j < 8; ++j)

{

mt\_gotoXY(x + i, y + j) ;

bool value ;

if (bc\_getbigCharPos(big, i, j, &value))

return -1 ;

if (value)

bc\_printA((char)ACS\_CKBOARD) ;

else

printf("%c", ' ');

}

mt\_setDefaultColorSettings() ;

return 0 ;

}

/// Устанавливает значение знакоместа "большого символа"

/// \param big

/// \param x - столбец

/// \param y - строка

/// \param value - значение

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

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

if ((x < 0) || (x > 7) || (y < 0) || (y > 7))

return -1 ;

if (value)

big[int(y / 4)] |= (1 << (8 \*(y % 4) + (7 - x))) ;

else

big[int(y / 4)] &= ~(1 << (8 \*(y % 4) + (7 - x))) ;

return 0 ;

}

/// Возвращает значение позиции в "большом символе"

/// \param big

/// \param x - столбец

/// \param y - строка

/// \param value

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

int bc\_getbigCharPos(unsigned int\* big, int x, int y, bool \*value){

if ((x < 0) || (x > 7) || (y < 0) || (y > 7))

return -1 ;

if (big[int(y / 4)] & (1 << (8 \*(y % 4) + (7 - x))))

\*value = true ;

else

\*value = false ;

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, unsigned int \* big, int count){

if (write(fd, big,count \* 2 \* sizeof(unsigned int)))

return -1 ;

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, unsigned int \* big, int need\_count, int \* count){

\*count = 0 ;

for (int i = 0; i < need\_count \* 2; ++i){

if (read(fd, &big[i], sizeof(unsigned int)) == -1)

return -1 ;

if (!((i + 1) % 2))

(\*count)++ ;

}

return 0 ;

}

**myReadkey.hpp**

#ifndef MYREADKEY\_HPP

#define MYREADKEY\_HPP

#include <termio.h>

#include <stdio.h>

#include <unistd.h>

extern termios save;

enum keys

{

KEY\_L,

KEY\_S,

KEY\_R,

KEY\_T,

KEY\_I,

KEY\_F5,

KEY\_F6 ,

KEY\_UP,

KEY\_DOWN,

KEY\_RIGHT,

KEY\_LEFT,

KEY\_ESC,

KEY\_ENTER,

KEY\_OTHER,

};

int rk\_readKey(enum keys \*key = NULL) ;

int rk\_myTermSave() ;

int rk\_myTermRestore() ;

int rk\_myTermRegime (bool regime, unsigned int vtime, unsigned int vmin, bool echo, bool sigint) ;

#endif //MYREADKEY\_HPP

**myReadkey.cpp**

#include "myReadkey.hpp"

termios save;

/// Возвращающую первую клавишу, которую нажал пользователь

/// \param key - Адрес переменной, в которую возвращается номер нажатой клавиши

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

int rk\_readKey (enum keys \* key){

fflush(stdout) ; // очистка потока вывода

char buffer[5] = "\0";

rk\_myTermRegime(0, 30, 0, 0, 0);

read(fileno(stdin), buffer, 5);

rk\_myTermRestore();

if (key == NULL)

return 0 ;

// int i = 0 ;

// while (buffer[i] != '\0'){

// printf("\n%d",(int)buffer[i]) ;

// i++ ;

// }

if (buffer[0] == '\033')

if (buffer[1] == '\0')

\*key = KEY\_ESC ;

else if (buffer[1] == '[')

if (buffer[2] == 'A' and buffer[3] == '\0')

\*key = KEY\_UP ;

else if (buffer[2] == 'B' and buffer[3] == '\0')

\*key = KEY\_DOWN ;

else if (buffer[2] == 'C' and buffer[3] == '\0')

\*key = KEY\_RIGHT ;

else if (buffer[2] == 'D' and buffer[3] == '\0')

\*key = KEY\_LEFT ;

else if (buffer[2] == '1' and buffer[3] == '5')

\*key = KEY\_F5 ;

else if (buffer[2] == '1' and buffer[3] == '7')

\*key = KEY\_F6 ;

else

\*key = KEY\_OTHER ;

else

\*key = KEY\_OTHER ;

else if (buffer[0] == '\n' and buffer[1] == '\0')

\*key = KEY\_ENTER ;

else

if ((buffer[0] == 'l' or buffer[0] == 'L') and buffer[1] == '\0')

\*key = KEY\_L ;

else if ((buffer[0] == 's' or buffer[0] == 'S') and buffer[1] == '\0')

\*key = KEY\_S ;

else if ((buffer[0] == 'r' or buffer[0] == 'R') and buffer[1] == '\0')

\*key = KEY\_R ;

else if ((buffer[0] == 't' or buffer[0] == 'T') and buffer[1] == '\0')

\*key = KEY\_T ;

else if ((buffer[0] == 'i' or buffer[0] == 'I') and buffer[1] == '\0')

\*key = KEY\_I ;

else

\*key = KEY\_OTHER ;

return 0 ;

}

/// Функция, сохраняющая текущие параметры терминала

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

int rk\_myTermSave(){

if (tcgetattr(fileno(stdin), &save))

return -1 ;

return 0 ;

}

/// Функция, восстанавливающая сохраненные параметры терминала

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

int rk\_myTermRestore(){

tcsetattr(fileno(stdin), TCSAFLUSH, &save) ;

return 0 ;

}

/// Функция, переключающая режим работы терминала (канонический / неканонический)

/// \param regime

/// \param vtime

/// \param vmin

/// \param echo

/// \param sigint

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

int rk\_myTermRegime(bool regime, unsigned int vtime, unsigned int vmin, bool echo, bool sigint){

struct termios curr ;

tcgetattr(fileno(stdin), &curr) ;

if(regime)

curr.c\_lflag |= ICANON ;

else{

curr.c\_lflag &= ~ICANON ;

sigint ? (curr.c\_lflag |= ISIG) : (curr.c\_lflag &= ~ISIG) ;

echo ? (curr.c\_lflag |= ECHO) : (curr.c\_lflag &= ~ECHO) ;

curr.c\_cc[VMIN] = vmin ;

curr.c\_cc[VTIME] = vtime ;

}

tcsetattr(0,TCSAFLUSH,&curr);

return 0;

}

**myUI.hpp**

#ifndef MYUI\_HPP

#define MYUI\_HPP

#include <string.h>

#include "myBigChars.hpp"

#include "SimpleComputer.hpp"

int ui\_update() ;

int ui\_setMCellValue() ;

int ui\_saveMemory() ;

int ui\_loadMemory() ;

#endif //MYUI\_HPP

**myUI.cpp**

#include "myUI.hpp"

#include "myReadkey.hpp"

int drawingBoxes() ;

int drawingTexts() ;

int drawingMemory() ;

int drawingFlags() ;

int drawingBigChar() ;

bool checkCorrectInput(const char buffer[10]) ;

int ui\_messageOutput(char \*str, enum colors color) ;

/// Обновление интерфейса пользователя

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

int ui\_update(){

mt\_clrScreen() ;

if (drawingBoxes())

return -1 ;

if (drawingTexts())

return -1 ;

if (drawingMemory())

return -1 ;

if (drawingFlags())

return -1 ;

if (drawingBigChar())

return -1 ;

mt\_gotoXY(1, 23) ;

printf("Input/Output:\n") ;

return 0 ;

}

int ui\_setMCellValue(){

char buffer[10] ;

printf("Set the value of the cell under the number \033[38;5;%dm0x%X\033[0m\n", colors::SOFT\_GREEN, currMemCell) ;

printf("Enter value in \033[38;5;%dmhex\033[0m format > ", colors::PEACH);

fgets(buffer, 10, stdin) ;

fflush(stdin) ; // очистка потока ввода

// printf("%d",strlen(buffer)) ;

if (!checkCorrectInput(buffer)){

ui\_messageOutput((char \*)"Invalid input", colors::RED) ;

return -1 ;

}

long int number ;

char \* tmp ;

// TODO: добавить проверки на допустимые значения

if (buffer[0] == '+') {

number = strtol(&buffer[1], &tmp, 16) ;

if(number > 0x3FFF){

ui\_messageOutput((char \*)"The command value must not exceed 14 bits (0x3FFF)", colors::RED) ;

return -1 ;

}

sc\_memorySet(currMemCell, (short int)number) ;

}

else{

number = strtol(buffer, &tmp, 16);

if(number > 0x3FFF){

ui\_messageOutput((char \*)"The value must not exceed 14 bits (0x3FFF)", colors::RED) ;

return -1 ;

}

number = (1 << 14) | number ;

sc\_memorySet(currMemCell, (short int)number) ;

}

return 0 ;

}

int ui\_saveMemory(){

char filename[101] ;

printf("Saving file...\n") ;

printf("Enter the file name to save > ");

mt\_setFGcolor(colors::SOFT\_GREEN) ;

fgets(filename, 101, stdin) ;

mt\_setDefaultColorSettings() ;

// printf("%d %d",strlen(filename), filename[strlen(filename) - 1] == '\0') ;

if (strlen(filename) == 100 and filename[99] != '\n')

printf("\033[38;5;%dmThe file name is too long. The length is trimmed to the first 100 characters.\033[0m\n", BLUE) ;

else

filename[strlen(filename) - 1] = '\0' ;

if (sc\_memorySave(filename)){

ui\_messageOutput((char \*)"Failed to save memory", colors::RED) ;

return -1 ;

}

else

ui\_messageOutput((char \*)"Successful saving", colors::GREEN) ;

return 0 ;

}

int ui\_loadMemory(){

char filename[101] ;

printf("Loading file...\n") ;

printf("Enter the file name to load > ");

mt\_setFGcolor(colors::SOFT\_GREEN) ;

fgets(filename, 101, stdin) ;

mt\_setDefaultColorSettings() ;

if (strlen(filename) == 100 and filename[99] != '\n'){

ui\_messageOutput((char \*)"The name of the file to open is too long (up to 100 characters are allowed)", colors::BLUE) ;

return -1 ;

}

filename[strlen(filename) - 1] = '\0' ;

if (sc\_memoryLoad(filename)){

ui\_messageOutput((char \*)"Failed to load memory", colors::RED) ;

return -1 ;

}

else

ui\_messageOutput((char \*)"Successful loading", colors::GREEN) ;

return 0 ;

}

/// 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, 52, 10)) // Окно BigChars

return -1 ;

if (bc\_box(53, 13, 31, 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(54,13) ;

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(54,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) ;

short int tmp ;

sc\_memoryGet(i \* 10 + j, &tmp) ;

if ((i \* 10 + j) == currMemCell)

mt\_setBGcolor(colors::GREEN) ;

if((tmp >> 14) & 1)

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

else

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

mt\_setDefaultColorSettings() ;

}

return 0 ;

}

/// Отрисовка флагов

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

int drawingFlags(){

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

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

bool value ;

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

return -1 ;

mt\_gotoXY(68 + (i \* 2), 11) ;

if (value){

mt\_setFGcolor(colors::BLUE) ;

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

}

else{

mt\_setFGcolor(colors::GRAY) ;

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

}

mt\_setDefaultColorSettings() ;

}

return 0 ;

}

/// Отрисовка "BigChar'ов"

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

int drawingBigChar(){

short int tmp ;

sc\_memoryGet(currMemCell, &tmp) ;

if(!((tmp >> 14) & 1))

bc\_printBigChar(bc[16], 2, 14, GREEN) ; // +

tmp = tmp & 0x3FFF ;

for (int i = 0; i < 4; ++i) {

int ch = (tmp & ( 0xF << (4 \* (3 - i)) )) >> (4 \* (3 - i)) ;

bc\_printBigChar(bc[ch], 2 + 8 \* (i + 1) + 2 \* (i + 1), 14, GREEN) ;

}

return 0 ;

}

bool checkCorrectInput(const char buffer[10]){

// printf("\n%d",strlen(buffer));

int i = 0 ;

if (buffer[0] == '+'){

if (strlen(buffer) == 2 or strlen(buffer) > 6)

return false ;

++i ;

}

else

if (strlen(buffer) == 1 or strlen(buffer) > 5)

return false ;

for (i; i < strlen(buffer) - 1; ++i)

// printf("\n%d", isxdigit(buffer[i])) ;

if (!(isxdigit(buffer[i])))

return false ;

return true ;

}

int ui\_messageOutput(char \*str, enum colors color){

printf("\033[38;5;%dm%s\033[0m", color, str) ;

rk\_readKey() ;

return 0 ;

}