

МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ  
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ  
СИСТЕМ

**ПРАКТИЧЕСКАЯ РАБОТА №4**

по дисциплине «Архитектура ЭВМ»

Тема: «Консоль управления моделью Simple Computer. Клавиатура.  
Обработка нажатий клавиш. Неканонический режим работы терминала.»

Выполнил: Наумов Алексей ИС-142

Проверил: ассистент кафедры ВС Курзин А.С.

Новосибирск 2023

## 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_mytermrestore (void)` - сохраняет текущие параметры терминала;
  - `int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)` - восстанавливает сохранённые параметры терминала.
  - `int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)` - переключает терминал между режимами. Для неканонического режима используются значения второго и последующего параметров.
4. Оформите разработанные функции как статическую библиотеку myReadkey. Подготовьте заголовочный файл для неё.

## Результат:

Ввод информации через терминал может осуществляться в одном из двух режимов: каноническом, в котором информация передаётся в ЭВМ только в виде законченных строк (т.е. после нажатия клавиши «ВВОД» или «ENTER»); и неканоническом, при котором вводимая информация сразу поступает в ЭВМ.

```
mt_clrscr ();
rk_mytermregime (1, 0, 1, 0, 1); // за выбор между режимом отвечает 1 передаваемая переменная 0-неканон 1-канон
sc_memoryInit ();
for (int i = 3; i < 100; i += 2)
```

Memory

+0000	+0000	+0000	+0006	+0000	+0010	+0000	+0014	+0000	+0018
+0000	+0022	+0000	+0026	+0000	+0030	+0000	+0034	+0000	+0038
+0000	+0042	+0000	+0046	+0000	+0050	+0000	+0054	+0000	+0058
+0000	+0062	+0000	+0066	+0000	+0070	+0000	+0074	+0000	+0078
+0000	+0082	+0000	+0086	+0000	+0090	+0000	+0094	+0000	+0098
+0000	+0102	+0000	+0106	+0000	+0110	+0000	+0114	+0000	+0118
+0000	+0122	+0000	+0126	+0000	+0130	+0000	+0134	+0000	+0138
+0000	+0142	+0000	+0146	+0000	+0150	+0000	+0154	+0000	+0158
+0000	+0162	+0000	+0166	+0000	+0170	+0000	+0174	+0000	+0178
+0000	+0182	+0000	+0186	+0000	+0190	+0000	+0194	+0000	+0198

accumulator

0312

instructionCounter

0000

Operation

+00 : 00

Flags

F D A O C

Keys

l - load  
s - save  
r - run  
t - step  
i - reset  
F5 - accumulator  
F6 - instructionCounter

Memory

+0000	+0000	+0000	+0006	+0000	+0010	+0000	+0014	+0000	+0018
+0000	+0022	+0000	+0026	+0000	+0030	+0000	+0034	+0000	+0038
+0000	+0042	+0000	+0046	+0000	+0050	+0000	+0054	+0000	+0058
+0000	+0062	+0000	+0066	+0000	+0070	+0000	+0074	+0000	+0078
+0000	+0082	+0000					+0094	+0000	+0098
+0000	+0102	+0000					+0114	+0000	+0118
+0000	+0122	+0000					+0134	+0000	+0138
+0000	+0142	+0000	+0146	+0000	+0150	+0000	+0154	+0000	+0158
+0000	+0162	+0000	+0166	+0000	+0170	+0000	+0174	+0000	+0178
+0000	+0182	+0000	+0186	+0000	+0190	+0000	+0194	+0000	+0198

Set memory value to

accumulator

0312

instructionCounter

0001

Operation

+00 : 00

Flags

F D A O C

Keys

l - load  
s - save  
r - run  
t - step  
i - reset  
F5 - accumulator  
F6 - instructionCounter

```
mt_clrscr ();
rk_mytermregime (0, 0, 1, 0, 1); // за выбор между режимом отвечает 1 передаваемая переменная 0-неканон 1-канон
sc_memoryInit ();
```

Memory

+0000	+0000	+0000	+0006	+0000	+0010	+0000	+0014	+0000	+0018
+0000	+0022	+0000	+0026	+0000	+0030	+0000	+0034	+0000	+0038
+0000	+0042	+0000	+0046	+0000	+0050	+0000	+0054	+0000	+0058
+0000	+0062	+0000	+0066	+0000	+0070	+0000	+0074	+0000	+0078
+0000	+0082	+0000	Set memory value to				+0094	+0000	+0098
+0000	+0102	+0000	234	<			+0114	+0000	+0118
+0000	+0122	+0000					+0134	+0000	+0138
+0000	+0142	+0000	+0146	+0000	+0150	+0000	+0154	+0000	+0158
+0000	+0162	+0000	+0166	+0000	+0170	+0000	+0174	+0000	+0178
+0000	+0182	+0000	+0186	+0000	+0190	+0000	+0194	+0000	+0198

accumulator

0312

instructionCounter

0001

Operation

+00 : 00

Flags

F D A O C

Keys

l - load  
s - save  
r - run  
t - step  
i - reset  
F5 - accumulator  
F6 - instructionCounter

## Main.c

```
#include "bc.h"
#include "readkey.h"
#include "prototype.h"
#include "term.h"
#include "term_gui.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int
main ()
{ //big перенести

    int big[18][2] = {
        { 0x4242423C, 0x3C424242 }, // 0
        { 0x48506040, 0x40404040 }, // 1
        { 0x2042423C, 0x7E020418 }, // 2
        { 0x3840423C, 0x3C424040 }, // 3
        { 0x7E424478, 0x40404040 }, // 4
        { 0x3E02027E, 0x3C424040 }, // 5
        { 0x3E02423C, 0x3C424242 }, // 6
        { 0x2040407E, 0x10101010 }, // 7
        { 0x3C42423C, 0x3C424242 }, // 8
        { 0x7C42423C, 0x3C424040 }, // 9
        { 0x66663C18, 0x66667E7E }, // A
        { 0x3E66663E, 0x3E666666 }, // B
        { 0x0202423C, 0x3C420202 }, // C
        { 0x4444443E, 0x3E444444 }, // D
        { 0x3E02027E, 0x7E020202 }, // E
        { 0x1E02027E, 0x02020202 }, // F
        { 0x7E181800, 0x0018187E }, // +
        { 0x00181800, 0x00181800 } // :
    };

    mt_clrscr ();
    rk_mytermregime (0, 0, 1, 0, 1); //за выбор между режимом отвечает 1 передаваемая
    переменная 0-неканон 1-канон
    sc_memoryInit ();
    for (int i = 3; i < 100; i += 2)
    {
        sc_memorySet (i, i * 2);
    }
    sc_regInit ();
    // for (int j=1; j<5; j++)//попробую через цикл
    // {
    //     sc_regSet (j, 1);
    // }
    sc_regSet (1, 1);
    sc_regSet (2, 1);
    sc_regSet (4, 1);
    sc_regSet (8, 1);
    sc_regSet (16, 1);
    sc_accumSet (312);
    sc_countSet (3);
    g_interface (*big);
    rk_mytermrestore ();
    return 0;
}
```

## Prototype.c

```
#include "prototype.h"

int sc_memory[MEMSIZE];
int sc_register;
int sc_accum;
int sc_count;

int
sc_memoryInit () //инициализирует массив из 100 элементов
{
    memset (sc_memory, 0, MEMSIZE * sizeof (sc_memory[0]));
    return 0;
}

int
sc_memorySet (int address,
              int value) // устанавливает значение блока памяти
{
    if (address < 0 || address >= MEMSIZE)
    {
        BIT_SET (sc_register, FLAG_WRONG_ADDRESS);
        return ERR_WRONG_ADDRESS;
    }
    sc_memory[address] = value;
    return 0;
}

int
sc_memoryGet (
    int address, // gets the value of [address] memory unit and
    int *value) //получает значение блока памяти и возвращает его в переменную
{
    if (address < 0 || address >= MEMSIZE)
    {
        BIT_SET (sc_register, FLAG_WRONG_ADDRESS);
        return ERR_WRONG_ADDRESS;
    }
    *value = sc_memory[address];
    return 0;
}

int
sc_memorySave (char *filename) //сохраняет память в бинарный файл
{
    FILE *f = fopen (filename, "wb");
    if (!f)
    {
        return 1;
    }
    fwrite (sc_memory, sizeof (int), sizeof (sc_memory) / sizeof (int), f);
    fclose (f);
    return 0;
}

int
sc_memoryLoad (char *filename) //загружает оперативную память из файла
{
    FILE *f = fopen (filename, "rb");
    if (!f)
    {
        return 1;
    }
}
```

```

fread (sc_memory, sizeof (int), sizeof (sc_memory) / sizeof (int), f);
fclose (f);
return 0;
}

int
sc_regInit (void) //инициализирует регистр флагов с 0
{
    sc_register = 0;
    return 0;
}

int
sc_regSet (
    int reg,
    int value) //устанавливает значение регистра флага #define-s используются
               //для номера регистров, если неверный номер регистра то ошибка
{
    if (reg < 0 || reg > 5)
    {
        return ERR_WRONG_FLAG;
    }
    if (!value)
    {
        BIT_DEL (sc_register, reg);
        return 0;
    }
    if (value != 1)
    {
        BIT_SET (sc_register, FLAG_OVERFLOW);
        return ERR_WRONG_VALUE;
    }
    BIT_SET (sc_register, reg);
    return 0;
}

int
sc_regGet (
    int reg,
    int *value) //получает значение флага, если неверный регистр то ошибка
{
    if (reg < 0 || reg > 5)
    {
        return ERR_WRONG_FLAG;
    }
    *value = BIT_GET (sc_register, reg);
    return 0;
}

int
sc_commandEncode (
    int command, int operand,
    int *value) //кодирует команду с определенным номером и операндом
               // помещает результат в значение, если он неправильный
               // или операнд - ошибка, значение не меняется.
{
    // commands list: 0x10, 0x11, 0x20, 0x21, 0x30, 0x31, 0x32, 0x33, 0x40, 0x41,
    // 0x42, 0x43, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x60,
    // 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x70, 0x71, 0x72,
    // 0x73, 0x74, 0x75, 0x76
    if ((command > 0x76) || (command < 0x10)
        || ((command > 0x11) & (command < 0x20))
        || ((command > 0x21) & (command > 0x30)))

```

```

    || ((command > 0x33) & (command < 0x40))
    || ((command > 0x43) & (command < 0x51)))
{
    sc_regSet (FLAG_WRONG_COMMAND, 1);
    return ERR_WRONG_COMMAND;
}
if (operand < 0 || operand > 127)
{
    sc_regSet (FLAG_WRONG_OPERAND, 1);
    return ERR_WRONG_OPERAND;
}
int encoded = 0b0000000000000000 | command;
encoded <<= 7;
encoded |= operand;
*value = encoded;
return 0;
}

int
sc_commandDecode (int value, int *command,
                  int *operand) // декодирует значение как команду sc, если
                                // декодирование невозможно устанавливает
                                // команду error и возвращает ошибку.
{
    if ((value & (1 << 14)) != 0)
    {
        sc_regSet (FLAG_WRONG_COMMAND, 1);
        return ERR_WRONG_COMMAND;
    }
    *command = (value >> 7);
    value -= (*command << 7);
    *operand = value;
    return 0;
}

int
sc_accumSet (int value)
{
    sc_accum = value;
    return 0;
}

int
sc_accumGet (int *value)
{
    *value = sc_accum;
    return 0;
}

int
sc_countSet (int value)
{
    sc_count = value;
    return 0;
}

int
sc_countGet (int *value)
{
    *value = sc_count;
    return 0;
}

```



## Prototype.h

```
#pragma once

#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MEMSIZE 100

// флаги
#define FLAG_WRONG_COMMAND 16
#define FLAG_WRONG_OPERAND 8
#define FLAG_WRONG_ADDRESS 4
#define FLAG_DIV_BY_ZERO 2
#define FLAG_OVERFLOW 1

// ошибки
#define ERR_WRONG_ADDRESS -1
#define ERR_WRONG_FLAG -2
#define ERR_WRONG_VALUE -3
#define ERR_WRONG_COMMAND -4
#define ERR_WRONG_OPERAND -5

// битовые операции
#define BIT_SET(X, Y) X = X | (1 << (Y - 1))
#define BIT_DEL(X, Y) X = X & (~(1 << (Y - 1)))
#define BIT_GET(X, Y) X >> (Y - 1) & 0x1

int sc_memoryInit (); // инициализация массива из 100 элементов

int sc_memorySet (int address,
                  int value); // устанавливает значение блока памяти

int sc_memoryGet (int address, int *value); // получает значение блока памяти и
// возвращает его в значение var

int sc_memorySave (char *filename); // сохраняет память в бинарный файл

int sc_memoryLoad (char *filename); // загружает оперативную память из файла

int sc_regInit (void); // инициализирует регистр флагов с нуля

int sc_regSet (
    int reg,
    int value); // устанавливает флаг значение регистра #define-s используется
// для номеров регистров если неверный номер - то ошибка.

int sc_regGet (
    int reg,
    int *value); // получает значение флага, если неправильный регистр то ошибка

int sc_commandEncode (
    int command, int operand,
    int *value); // кодирует команду с определенным номером и операндом
// помещает результат в значение если он неправильная команда
// или операнд - ошибка, значение не меняется

int sc_commandDecode (int value, int *command,
                      int *operand); // декодирует значение как команду sc если
// декодирование невозможно устанавливает
// команду ошибки и возвращает ошибку.
```

```
int sc_accumGet (int *value);  
int sc_accumSet (int value);  
int sc_countSet (int value);  
int sc_countGet (int *value);
```

## bc.c

```
#include "bc.h"

int
bc_printA (char *str)
{
    ssize_t len = strlen (str) * sizeof (char);
    write (STDOUT_FILENO, "\E(0", 3);
    if (write (STDOUT_FILENO, str, len) != len)
    {
        return -1;
    }
    write (STDOUT_FILENO, "\E(B", 3);
    return 0;
}

int
bc_box (int x1, int y1, int x2, int y2)
{
    for (int i = 0; i < x2; i++)
    {
        for (int j = 0; j < y2; j++)
        {
            mt_gotoXY (x1 + i, y1 + j);
            if (i == 0 && j == 0)
            {
                bc_printA ("l");
            }
            else if (i == 0 && j == y2 - 1)
            {
                bc_printA ("k\n");
            }
            else if (i == x2 - 1 && j == 0)
            {
                bc_printA ("m");
            }
            else if (i == x2 - 1 && j == y2 - 1)
            {
                bc_printA ("j\n");
            }
            else if ((i == 0 || i == x2 - 1)
                     && (j > 0 && j < y2)) // horizontal line
            {
                bc_printA ("q");
            }
            else if ((i > 0 && i < x2)
                     && (j == 0 || j == y2 - 1)) // vertical line
            {
                bc_printA ("x");
            }
            else
            {
                write (STDOUT_FILENO, " ", sizeof (char));
            }
        }
    }
    return 0;
}

int
bc_printbigchar (int *big, int x, int y, enum colors fgcolor,
                 enum colors bgcolor)
{

```

```

mt_setbgcolor (bgcolor);
mt_setfgcolor (fgcolor);
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 8; j++)
    {
        mt_gotoXY (x + i, y + j);
        short value;
        bc_getbigcharpos (big, i, j, &value);
        if (!value)
        {
            write (STDOUT_FILENO, "", sizeof (char));
        }
        else
        {
            bc_printA ("a");
        }
    }
    write (STDOUT_FILENO, "\n", sizeof (char));
}
mt_setbgcolor (GREY);
mt_setfgcolor (LIGHT_GREY);
return 0;
}

int
bc_setbigcharpos (int *big, int x, int y, short int value)
{
    if (x < 0 || x > 7 || y < 0 || y > 7 || value > 1 || value < 0)
    {
        return 1;
    }
    int part = x / 4;
    x %= 4;
    if (value)
    {
        big[part] |= (1 << (8 * x + y));
    }
    else
    {
        big[part] &= ~(1 << (8 * x + y));
    }
    return 0;
}

int
bc_getbigcharpos (int *big, int x, int y, short int *value)
{
    if (x < 0 || x > 7 || y < 0 || y > 7)
    {
        return -1;
    }
    short int part = x / 4;
    x %= 4;
    if (big[part] & (1 << (8 * x + y)))
    {
        *value = 1;
    }
    else
    {
        *value = 0;
    }
    return 0;
}

```

```
}

int
bc_bigcharwrite (int fd, int *big, int count)
{
    for (int i = 0; i < count * 2; i++)
    {
        if (write (fd, &big[i], sizeof (int)) == -1)
        {
            return 1;
        }
    }
    return 0;
}

int
bc_bigcharread (int fd, int *big, int need_count, int *count)
{
    for (*count = 0; (*count < need_count * 2); *count += 1)
    {
        if (read (fd, &big[*count], sizeof (int)) == -1)
        {
            return 1;
        }
    }
    return 0;
}
```

## bc.h

```
#pragma once

#include "term.h"

int bc_printA (char *str);
int bc_box (int x1, int y1, int x2, int y2);
int bc_printbigchar (int *big, int x, int y, enum colors fgcolor,
                    enum colors bgcolor);
int bc_setbigcharpos (int *big, int x, int y, short int value);
int bc_getbigcharpos (int *big, int x, int y, short int *value);
int bc_bigcharwrite (int fd, int *big, int count);
int bc_bigcharread (int fd, int *big, int need_count, int *count);
```

## term.c

```
#include "term.h"

int
mt_clrscr (void) // очищает экран и перемещает курсор в верхний левый угол
{
    if (write (STDOUT_FILENO, CLEAR, strlen (CLEAR))
        < sizeof (char) * strlen (CLEAR))
    {
        return -1;
    }
    return 0;
}

int
mt_gotoXY (
    int x,
    int y) // перемещает курсор к введенным координатам (x, y) = (row, col)
{
    char go[30];
    sprintf (go, "\E[%d;%dH", x, y);
    if (write (STDOUT_FILENO, go, strlen (go)) < sizeof (char) * strlen (go))
    {
        return -1;
    }
    return 0;
}

int
mt_getscreensize (int *rows, int *cols) // получает размер экрана терминала
                                         // (количество строк и столбцов)
{
    struct winsize ws;
    if (ioctl (1, TIOCGWINSZ, &ws))
    {
        return -1;
    }
    *rows = ws.ws_row;
    *cols = ws.ws_col;
    return 0;
}

int
mt_setfgcolor (enum colors color) // устанавливает цвет фона для всех строк и
                                   // столбцов всего терминала
{
    char foreground[30];
    sprintf (foreground, "\E[38;5;%dm", color);
    if (write (STDOUT_FILENO, foreground, strlen (foreground))
        < sizeof (char) * strlen (foreground))
    {
        return -1;
    }
    return 0;
}

int
mt_setbgcolor (enum colors color) // устанавливает цвет фона только для
                                   // предстоящих символов
{
    char background[30];
    sprintf (background, "\E[48;5;%dm", color);
    if (write (STDOUT_FILENO, background, strlen (background))
```

```

    < sizeof (char) * strlen (background))
{
    return -1;
}
return 0;
}

```

## term.h

```
#pragma once

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>

#define CLEAR "\E[H\E[2J"
#define RESETCOLOR "\033[0m"
#define RESETCOLORF "\033[39m"

enum colors
{
    LIGHT_GREY = 247,
    GREY = 237,
    BLACK = 16,
    RED = 124,
    GREEN = 41,
    BLUE = 20,
    YELLOW = 184,
};

int mt_clrscr (void); // очищает экран и перемещает курсор в верхний левый угол
int mt_gotoXY (
    int x,
    int y); // перемещает курсор к введенным координатам (x, y) = (row, col)
int mt_getscreenize (int *rows, int *cols); //получает размер экрана терминала
// (количество строк и столбцов)
int mt_setfgcolor (enum colors color); // устанавливает цвет фона для всех и
// столбцов всего терминала
int mt_setbgcolor (enum colors color); // устанавливает цвет фона только для
// предстоящих символов
```



## term\_gui.c

```
#include "term_gui.h"
#include "bc.h"
#include "prototype.h"
#include "term.h"
#include <math.h>
#include <stdlib.h>
#include <string.h>

int
g_flags (char **val)
{
    int flag = 0;
    sc_regGet (1, &flag);
    char F = flag == 1 ? 'F' : ' ';
    sc_regGet (2, &flag);
    char D = flag == 1 ? 'D' : ' ';
    sc_regGet (4, &flag);
    char A = flag == 1 ? 'A' : ' ';
    sc_regGet (8, &flag);
    char O = flag == 1 ? 'O' : ' ';
    sc_regGet (16, &flag);
    char C = flag == 1 ? 'C' : ' ';
    char buff[14];
    sprintf (buff, "%c %c %c %c %c", F, D, A, O, C);
    *val = buff;
    return 0;
}

int
g_static ()
{
    bc_box (1, 1, 12, 63);
    mt_gotoXY (0, 28);
    write (STDOUT_FILENO, " Memory ", 8 * sizeof (char));
    bc_box (1, 64, 3, 39);
    mt_gotoXY (1, 77);
    write (STDOUT_FILENO, " accumulator ", 13 * sizeof (char));
    bc_box (4, 64, 3, 39);
    mt_gotoXY (4, 73);
    write (STDOUT_FILENO, " instructionCounter ", 20 * sizeof (char));
    bc_box (7, 64, 3, 39);
    mt_gotoXY (7, 78);
    write (STDOUT_FILENO, " Operation ", 11 * sizeof (char));
    bc_box (10, 64, 3, 39);
    mt_gotoXY (10, 79);
    write (STDOUT_FILENO, " Flags ", 7 * sizeof (char));
    bc_box (13, 1, 12, 63);
    bc_box (13, 64, 12, 39);
    mt_gotoXY (13, 67);
    write (STDOUT_FILENO, " Keys ", 7 * sizeof (char));
    char *str[7] = { "l - load",
                    "s - save",
                    "r - run",
                    "t - step",
                    "i - reset",
                    "F5 - accumulator",
                    "F6 - instructionCounter" };
    for (int i = 0; i < 7; i++)
    {
        mt_gotoXY (15 + i, 66);
        write (STDOUT_FILENO, str[i], strlen (str[i]));
    }
}
```

```

    mt_gotoXY (33, 0);
    return 0;
}

int
g_memorybox ()
{
    int k = 0;
    for (int i = 2; i < 12; i++)
    {
        for (int j = 3; j < 63; j += 6)
        {
            mt_gotoXY (i, j);
            char buff[6];
            int val;
            sc_memoryGet (k++, &val);
            sprintf (buff, "+%04d", val);
            write (STDERR_FILENO, buff, 6 * sizeof (char));
        }
    }
    return 0;
}

int
g_accumbox ()
{
    mt_gotoXY (2, 80);
    char buff[5];
    int val;
    sc_accumGet (&val);
    sprintf (buff, "%04d", val);
    write (STDOUT_FILENO, buff, 5 * sizeof (char));
    mt_gotoXY (33, 0);
    return 0;
}

int
g_counterbox ()
{
    mt_gotoXY (5, 80);
    char buff[5];
    int val;
    sc_countGet (&val);
    sprintf (buff, "%04d", val);
    write (STDOUT_FILENO, buff, 5 * sizeof (char));
    mt_gotoXY (33, 0);
    return 0;
}

int
g_operationbox ()
{
    mt_gotoXY (8, 79);
    write (STDOUT_FILENO, "+00 : 00", 8 * sizeof (char));
    mt_gotoXY (33, 0);
    return 0;
}

int
g_flagbox ()
{
    mt_gotoXY (11, 79);
    char *val;

```

```

g_flags (&val);
write (STDOUT_FILENO, val, 14 * sizeof (char));
mt_gotoXY (33, 0);
return 0;
}

int
g_bcbox (int *big)
{
    int count = 0;
    int val = 0;
    sc_countGet (&count);
    sc_memoryGet (count, &val);
    int digit[2] = { big[2 * 16], big[2 * 16 + 1] };
    bc_printbigchar (digit, BC_X, BC_START, GREEN, GREY);
    for (int i = 3; i >= 0; i--)
    {
        int radix = (int)pow (10, i);
        int k = 0;
        switch ((val / radix) % 10)
        {
            case 0x0:
                k = 0;
                break;
            case 0x1:
                k = 1;
                break;
            case 0x2:
                k = 2;
                break;
            case 0x3:
                k = 3;
                break;
            case 0x4:
                k = 4;
                break;
            case 0x5:
                k = 5;
                break;
            case 0x6:
                k = 6;
                break;
            case 0x7:
                k = 7;
                break;
            case 0x8:
                k = 8;
                break;
            case 0x9:
                k = 9;
                break;
            case 0xA:
                k = 10;
                break;
            case 0xB:
                k = 11;
                break;
            case 0xC:
                k = 12;
                break;
            case 0xD:
                k = 13;
                break;
        }
    }
}

```

```

        case 0xE:
            k = 14;
            break;
        case 0xF:
            k = 15;
            break;
        default:
            k = 17;
            break;
    }
    int digit[2] = { big[2 * k], big[2 * k + 1] };
    bc_printbigchar (digit, BC_X, BC_START + (4 - i) * BC_STEP, GREEN, GREY);
}
mt_gotoXY (33, 0);
return 0;
}

```

## term\_gui.h

```

#pragma once

#include <sys/param.h>

#define BC_X 15
#define BC_START 8
#define BC_STEP 8
#define X_START 2
#define Y_START 3
#define Y_STEP 6

int g_memorybox ();
int g_accumbox ();
int g_counterbox ();
int g_operationbox ();
int g_flagbox ();
int g_bcbox (int *big);
int g_flags (char **val);
int g_static ();

```

## Makefile

```

DIRGUARD = @mkdir -p $(@D)

all: bin/simplecomputer
.PHONY: bin/simplecomputer
bin/simplecomputer: src/main.c bin/lib.a
$(DIRGUARD)
gcc -Wall -Wextra -I src -o $@ $^ -lm

bin/lib.a: obj/bc.o obj/prototype.o obj/term.o obj/term_gui.o
$(DIRGUARD)
ar rcs $@ $^

obj/%.o: src/%.c
$(DIRGUARD)
gcc -Wall -Wextra -c -o $@ $< -lm

test: bin/test
.PHONY: bin/test
bin/test: test/*.c bin/lib.a
$(DIRGUARD)
gcc -Wall -Wextra -I src -MMD -I thirdparty -o $@ $^ -lm

```

## readkey.h

```
#pragma once

enum keys
{
    KEY_S,
    KEY_L,
    KEY_R,
    KEY_T,
    KEY_I,
    KEY_F5,
    KEY_F6,
    KEY_UP,
    KEY_DOWN,
    KEY_RIGHT,
    KEY_LEFT,
    KEY_ENTER,
    KEY_ESCAPE,
    KEY_DEFAULT,
};

int rk_readkey (enum keys *key);
int rk_mytermsave (void);
int rk_mytermrestore (void);
int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint);
```

## readkey.c

```
#include "readkey.h"
#include "term.h"
#include <string.h>
#include <termios.h>
#include <unistd.h>

struct termios current, backup;

int
rk_readkey (enum keys *key)
{
    char buff[16];
    int numRead;
    if (tcgetattr (STDOUT_FILENO, &backup) != 0)
    {
        return -1;
    }
    if (rk_mytermregime (0, 0, 1, 0, 1) != 0)
    {
        return -1;
    }
    numRead = read (STDOUT_FILENO, buff, 15);
    buff[numRead] = '\0';
    if (strcmp (buff, "l") == 0)
    {
        *key = KEY_L;
    }
    else if (strcmp (buff, "s") == 0)
    {
        *key = KEY_S;
    }
    else if (strcmp (buff, "r") == 0)
    {
        *key = KEY_R;
    }
}
```

```

else if (strcmp (buff, "q") == 0)
{
    *key = KEY_ESCAPE;
}
else if (strcmp (buff, "t") == 0)
{
    *key = KEY_T;
}
else if (strcmp (buff, "i") == 0)
{
    *key = KEY_I;
}
else if ((strcmp (buff, "\n") == 0) || (strcmp (buff, "\r") == 0))
{
    *key = KEY_ENTER;
}
else if ((strcmp (buff, "\E[15~") == 0) || (strcmp (buff, "\E[[E") == 0))
{
    *key = KEY_F5;
}
else if (strcmp (buff, "\E[17~") == 0)
{
    *key = KEY_F6;
}
else if (strcmp (buff, "\E[A") == 0)
{
    *key = KEY_UP;
}
else if (strcmp (buff, "\E[B") == 0)
{
    *key = KEY_DOWN;
}
else if (strcmp (buff, "\E[C") == 0)
{
    *key = KEY_RIGHT;
}
else if (strcmp (buff, "\E[D") == 0)
{
    *key = KEY_LEFT;
}
if (tcsetattr (STDOUT_FILENO, TCSANOW, &backup) != 0)
{
    return -1;
}
return 0;
}

int
rk_mytermsave ()
{
    return tcsetattr (STDOUT_FILENO, TCSANOW, &current);
}

int
rk_mytermrestore ()
{
    return tcsetattr (STDOUT_FILENO, TCSANOW, &backup);
}

int
rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)
{
    if (tcgetattr (STDOUT_FILENO, &current))

```

```
{
    return 1;
}
if (regime)
{
    current.c_lflag |= ICANON;
}
else
{
    current.c_lflag &= ~ICANON;
}
if (echo)
{
    current.c_lflag |= ECHO;
}
else
{
    current.c_lflag &= ~ECHO;
}

if (sigint)
{
    current.c_lflag |= ISIG;
}
else
{
    current.c_lflag &= ~ISIG;
}
current.c_cc[VTIME] = vtime;
current.c_cc[VMIN] = vmin;
return tcsetattr (STDOUT_FILENO, TCSANOW, &current);
}
```