МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

Лабораторная работа №2
ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ (ТЕКСТОВЫЙ РЕЖИМ)

Студентка гр. 6306 _____ Пустовойтова А. А.

Преподаватель _____ Бекенева Я. А.

Санкт-Петербург
2017

Краткие сведения

Видеорежимы и их характеристика

Интегральной характеристикой особенностей работы адаптера является совокупность поддерживаемых им режимов. Поведение адаптера в том или ином режиме является фактическим стандартом и полностью характеризует все особенности адаптера, доступные для программиста средства управления адаптером и т.п. Режимы принято нумеровать, начиная с нуля. Чем совершеннее видеоадаптер, тем больше режимов он поддерживает. Как правило, более совершенные адаптеры полностью совместимы со своими предшественниками и с точки зрения прикладной программы отображает информацию точно так же, как и его предшественник. Некоторые режимы работы видеоадаптеров описаны в табл. 2.1.

Табл. 2.1. Режимы работы видеоадаптеров

Режим	Тип	Размер шрифта	Максимальное число страниц	Разрешение		Адрес видеобуфера	Тип Видеоадаптера
				графика	Текст		
0,1	Текст	8x8	8	_	40x25	B8000h	CGA, EGA, VGA,MCGA
	Текст	8x14	8	_	40x25	B8000h	EGA,VGA
	Текст	8x16	8	-	40x25	B8000h	MCGA
	Текст	9x16	8	_	40x25	B8000h	VGA
2,3	Текст	8x8	4	-	80x25	B8000h	CGA
	Текст	8x8	8	-	80x25	B8000h	EGA,VGA
	Текст	8x8	8	_	80x43	B8000h	EGA
	Текст	8x8	8	_	80x50	B8000h	VGA
	Текст	8x14	8	-	80x25	B8000h	EGA,VGA
	Текст	8x16	8		80x25	B8000h	MCGA
	Текст	9x16	8	-	80x25	B8000h	VGA
4,5	Граф.	8x8	1	320x200	40x25	B8000h	CGA,EGA, VGA,AT&T MCGA
6	Граф.	8x8	1	640x200	80x25	B8000h	EGA,VGA
7	Текст	8x14	4		80x25	B0000h	EGA,VGA
Dh	Граф.	8x8	8	320x200	40x25	A0000h	EGA,VGA

Eh	Граф.	8x8	4	640x200	80x25	A0000h	EGA,VGA
Fh	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
10h	Граф.	8x14	2	640x350	80x25	A0000h	EGA,VGA
llh	Граф.	8x16	1	640x480	80x30	A0000h	MCGA, VGA
12h	Граф.	8x16	1	640x480	80x30	A0000h	VGA

При всем многообразии режимов работы видеоадаптеров их можно объединить в две группы: текстовые и графические. Переключение из текстового режима в графический и наоборот означает полное изменение логики работы видеоадаптера с видеобуфером. Если видеоадаптер включен в текстовый режим, он рассматривает экран как совокупность так называемых текселов (texel - Text Element) (рис. 2.1).

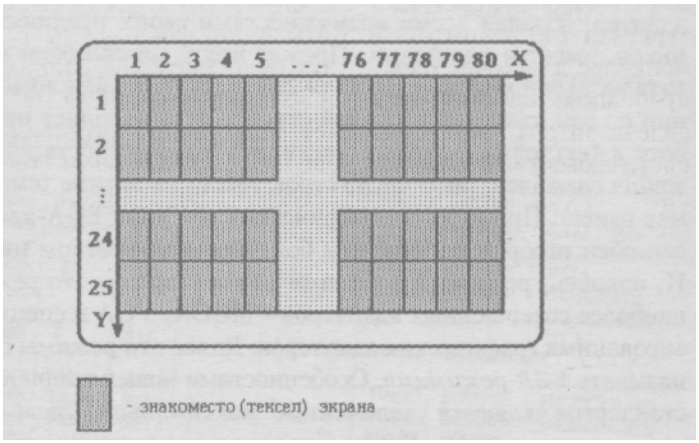


Рис. 2.1. Представление экрана в текстовых режимах "25 строк х 80 столбцов"

Каждому знакоместу экрана (текселу) в текстовом режиме соответствуют два байта памяти видеобуфера. Байт по четному адресу хранит ASCII-код символа, а следующий за ним байт по нечетному адресу кодирует особенности отображения символа на экране: цвет пикселов, из которых формируется очертание символа (Foreground Color), цвет всех остальных пикселов знакоместа или цвет фона символа (Background Color), мерцание символа и необходимость повышения яркости символа при отображении. Этот байт называется байтом атрибута. Закрепление битов байта атрибута приведено на рис. 2.2.

R, G, B -соответственно красный (Red), зеленый (Green), синий (Blue) цвета
(1 - цвет включен; 0 - цвет выключен)
Рис. 2.2. Назначение битов байта атрибута

Задавая различные числовые значения байту атрибута в видеобуфере, можно управлять цветом символов и цветом фона, на котором эти символы отображаются. Например, если значение байта атрибута равно 112, то выводится немерцающий символ черного цвета на сером фоне. Действительно, биты RGB цвета символа для данного кода атрибута равны нулю. Биты цвета фона равны 1, и на мониторе для точек фона будут смешиваться в необходимых пропорциях красный, синий и зеленый цвета. Для цветного видеоадаптера - это серый цвет. Повышение интенсивности цвета символа выполняется путем установки бита с номером 3 в 1. Светло-серый цвет - это белый цвет, поэтому на экране цветного монитора при работе видеоадаптера в текстовом режиме могут быть белые буквы, но не может быть белый фон. Например, символы, код атрибута которых в видеопамяти равен 15, будут отображаться белыми пикселами на черном фоне. В принципе, если задать цвета фона и символа одинаковыми, символы будут невидимыми, например красный символ на красном фоне (атрибут 0x44), что можно использовать в адаптерах, у которых мерцание символа с помощью бита 7 не реализовано.
Видеоадаптеры типов EGA и VGA имеют некоторые особенности использования бита интенсивности, которые будут рассмотрены несколько позже.
Видеопамять адаптера при работе в текстовых режимах доступна непосредственно из программы. Это значит, что любая ячейка видеобуфера может быть прочитана программой так же, как и обычная ячейка оперативной памяти. И как в обычную ячейку памяти, в видеобуфер возможна запись значений из программы. Адреса ячеек видеопамяти начинаются для разных типов адаптеров с разных границ, приведенных в табл. 2.1. Если адаптер работает в текстовых режимах "40 столбцов x 25 строк", то для хранения полного образа экрана (видеостраницы) требуется 25 x 40 x 2 = 2000 байт видеопамяти. В режимах "80 столбцов x 25 строк" видеостраница занимает уже 25 x 80 x 2 = 4000 байт.
Минимальная конфигурация видеоадаптера CGA имеет обычно 16K байт видеопамяти, что позволяет хранить 8 страниц текста в режимах 0 или 1 и 4 страницы в режимах 2 или 3.
Вывод на монитор содержимого видеобуфера происходит начиная с некоторого начального адреса, называемого смещением до видеостраницы. Страница 0 имеет нулевое смещение. Страница 1 в режиме "80 строк x 25 столбцов" начинается с адреса, смещенного на 4096 байт (l000h) относительно начального адреса видеопамяти, страница 2 - со смещения 8192 байт (2000h) и т.д. Если изменить значение смещения, произойдет переключение страницы, т.е. на экране возникнет образ другой страницы видеопамяти. Иногда переключение видеостраниц в текстовом режиме используется для реализации динамических изображений.
Видеоадаптер при работе в текстовом режиме периодически считывает содержимое ячеек видеобуфера и по коду символа и байту атрибута формирует пикселы, образующие в совокупности очертание символа и его фон. При этом байт символа служит индексом для входа в специальную таблицу - так называемую таблицу знакогенератора. Она содержит информацию, по которой видеоадаптер формирует пикселы для изображения того или иного символа. Число строк и столбцов в одной ячейке таблицы различно для различных

типов видеоадаптеров. Чем больше строк и столбцов использовано для символа, тем более качественно он изображается на экране.
Число знакомест в одной текстовой строке зависит от видеоадаптера и от режима его работы.
Переключение адаптера в один из графических режимов полностью изменяет логику работы аппаратуры видеосистемы. При работе в графическом режиме появляется возможность управлять цветом любой телевизионной точки экрана или пиксела. Число строк пикселов и число пикселов в каждой строке зависит от режима работы видеоадаптера. Таким образом, экран в графическом режиме представляет собой матрицу пикселов (рис. 2.3).

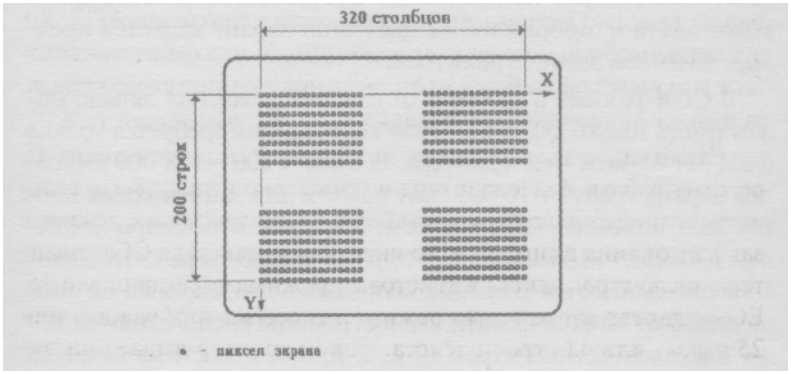


Рис. 2.3. Представление экрана в графических режимах 4,5 (320 столбцов x 200 строк)

Работа с текстовой информацией

Вывод информации на экран персонального компьютера может выполняться на трех уровнях:

1. на уровне MS-DOS с использованием функций прерывания 21h
2. на уровне BIOS с использованием функций прерывания 10h
3. непосредственным доступом к аппаратным средствам.

Вывод информации на уровне MS-DOS - мобильный, но самый медленный. Функции MS-DOS для вывода информации на экран вызывают драйвер консоли (выполняют вывод в специальный символьный файл CON). Если в системе инсталлирован специальный драйвер (например, ANSI.SYS), могут использоваться дополнительные средства по управлению экраном. Суть расширенного управления состоит в передаче драйверу консоли специальных управляющих строк. Драйвер опознает начало управляющей строки по символу ASCII с кодом 27 (1Bh). Передаваемые на экран вслед за ним символы рассматриваются как параметры команды, которую выполняет драйвер, например, перемещает курсор, устанавливает цвет символа и т.п. Сами управляющие символы не отображаются на экране. Таким образом, использование функций MS-DOS позволяет осуществить вывод через драйвер. Другие достоинства функций MS-DOS - автоматическое позиционирование курсора и скроллинг экрана, реакция на нажатие комбинации клавиш Ctrl-Break. Недостатком является невозможность непосредственного управления курсором и атрибутом символов. На уровне MS-DOS работают функции стандартного вывода, а их прототипы содержатся в файле <stdio.h>.
Вывод на уровне BIOS дает более широкие возможности по управлению экраном. Именно эти функции используются драйверами MS-DOS для вывода информации на экран. Недостатком функций BIOS является невысокая скорость вывода, что особенно заметно при работе в графических режимах. На уровне BIOS работают функции консольного вывода, а их прототипы помещены в файле <conio.h>.
Для приложений, критичных по скорости вывода, приходится выполнять вывод, используя непосредственный доступ к портам и видеопамяти адаптера. Такой способ позволяет достичь максимально возможной скорости вывода, но требует максимальных затрат труда программиста. Функции консольного вывода Turbo C могут по выбору пользователя работать и на самом нижнем уровне, выполняя доступ к видеобуферу при работе в текстовом режиме.

Функции

C++ включает большой набор функций ввода-вывода информации в окно экрана. Прототипы этих функций помещены в заголовочном файле <conio.h>. В отличие от функций стандартного ввода-вывода они позволяют управлять цветом выводимых символов и не пересекают пределы активного в данный момент окна. При достижении правой вертикальной границы курсор автоматически переходит на начало следующей строки в пределах окна, а при достижении нижней горизонтальной границы выполняется скроллинг окна вверх.

Функция clreol() стирает в текстовом окне строку, на которую установлен курсор, начиная с текущей позиции курсора и до конца строки (до правой вертикальной границы окна).

Функция clrscr() очищает все текстовое окно. Цвет "заливки" окна при очистке будет соответствовать значению, установленному символической переменной attribute в описании окна (структурная переменная по шаблону text_info). Функции управления цветом фона и символа описаны далее.

Функция delline() стирает в текстовом окне всю строку текста, на которую установлен курсор.

Функция insline() вставляет пустую строку в текущей позиции курсора со сдвигом всех остальных строк окна на одну строку вниз. При этом самая нижняя строка текста окна теряется.

Функция cprintf(const char *format,...) выполняет вывод информации с преобразованием по заданной форматной строке, на которую указывает format. Является аналогом функции стандартной библиотеки printf (), но выполняет вывод в пределах заданного окна. В отличие от printf () функция cprintf () иначе реагирует на специальный символ '\n': курсор переводится на новую строку, но не возвращается к левой границе окна. Поэтому для перевода курсора на начало новой строки текстового окна следует вывести последовательность символов CR-LF (0x0d, 0x0a). Остальные специальные символы воздействуют на курсор так же, как и в случае функций стандартного ввода-вывода. Функция возвращает число выведенных байтов, а не число обработанных полей, как это делает функция printf () .

Функция cputs(char *str) выводит строку символов в текстовое окно, начиная с текущей позиции курсора. На начало выводимой ASCII-строки указывает указатель str. Является аналогом функции стандартной библиотеки puts (), но выполняет вывод в пределах заданного окна и при выводе не добавляет специальный символ '\n'. Реакция cputs() на специальный символ '\n' аналогична реакции cprintf(). Функция возвращает ASCII-код последнего выведенного на экран символа. В отличие от puts() в функции отсутствует возврат символа EOF. Другими словами, вывод происходит на экран в любом случае.

Функция movetext(int left, int top, int right, int bottom,int destleft, int desttop) переносит окно, заданное координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов, в другое место на экране, заданное координатами левого верхнего угла нового положения окна. Размеры окна по горизонтали и вертикали сохраняются. Все координаты задаются относительно координат верхнего левого угла экрана (1,1). Функция возвращает ненулевое значение, если перенос заданного окна выполнен. В противном случае возвращается 0. Функция корректно выполняет перекрывающиеся переносы, т.е. переносы, в которых прямоугольная область-источник и область, в которую окно переносится, частично покрывают друг друга.

Функция putch(int ch) выводит символ в текущей позиции текстового окна экрана. Как и для функций cprintf(), cputs(), специальный символ ‘\n' вызывает только переход курсора на следующую строку текстового окна без возврата к его левой вертикальной границе. Остальные специальные символы воздействуют на курсор так же, как и для функций стандартного ввода-вывода. \

Функция puttext(int left, int top, int right, int bottom,void *source) выводит на экран текстовое окно, заданное координатами левого верхнего (left, top) и правого нижнего (right, bottom) углов. Символы и атрибуты располагаются в буфере, адрес начала которого задает указатель source. Другими словами, функция "открывает" (восстанавливает) текстовое окно экрана. Обычно используется вместе с функцией gettext(), выполняющей обратную операцию - запись в буфер source символов/атрибутов, полностью описывающих все знакоместа текстового окна. Функция проверяет по заданным координатам окна, можно ли

построить окно на экране для текущего режима видеоадаптера и корректны ли эти координаты. В случае, когда окно успешно выведено, возвращается ненулевое значение.
Функции highvideo (void), lowvideo (void) и normvideo (void) задают соответственно использование повышенной, пониженной и нормальной яркости для последующего вывода символов на экран.
Описываемые далее функции управляют атрибутом символа. Как отмечено ранее, атрибут задает битами 0-2 код цвета символа, бит 3 определяет повышение яркости, биты 4-6 задают код цвета фона символа, бит 7 определяет наличие или отсутствие мерцания символа. Возможно задать атрибут полностью либо задать только цвет символа или фона. Цвета могут задаваться либо числом, либо с использованием символических констант, значения которых определяет перечислимый тип COLORS (табл. 2.3):

Табл. 2.3.

```
Enum COLORS { /* Цвета нормальной яркости: */
    BLACK,          /* черный,              0    */
    BLUE,           /* синий,               1    */
    GREEN           /* зеленый              2    */
    CYAN            /* сине-зеленый             3
    RED,            /* красный,             4    */
    MAGENTA         /* красно-синий         5    */
    BROWN           /* коричневый           6    */
    LIGHTGRAY,      /* светло-серый.        7    */
                /* Цвета повышенной яркости: */
    DARKGRAY,       /* темно-серый,             8    */
    LIGHTBLUE,      /* ярко-синий,              9    */
    LIGHTGREEN      /* ярко-зеленый             10   */
    LIGHTCYAN,      /* яркий сине-зеленый,      11   */
    LIGHTRED,       /* ярко-красный,            12   */
    LIGHTMAGENTA,   /* яркий красно-синий,      13   */
    YELLOW,         /* желтый,                  14   */
    WHITE           /* белый.                   15
```

Яркие цвета могут задаваться только цвету символа. Кроме того, 7-й бит (бит мерцания) может быть задан как непосредственно в коде байта атрибута, так и с использованием символической константы BLINK, определяемой как код 128. Следует отметить тот факт, что если для цвета фона выбираются цвета с кодами 8-15, это устанавливает в единицу бит мерцания символа в байте атрибута.
Функция textattr(int newattr) устанавливает атрибут для функций, работающих с текстовыми окнами. Атрибут хранится в поле attribute структурной переменной по шаблону text_info, доступной через функцию gettextinfo(). Задаваемый атрибут может быть или числом, например 0x70 - атрибут инверсного изображения (черные символы на светло-сером фоне) или формироваться из символических констант, значения которых задает тип COLORS. Например, для задания мерцающих ярко-красных символов на сером фоне атрибут можно сформировать следующим образом:

```
BLINK | (BLACK << 4) | LIGHTRED
```

Тот же результат может быть получен и так:

```
(DARKGRAY << 4)| LIGHTRED
```

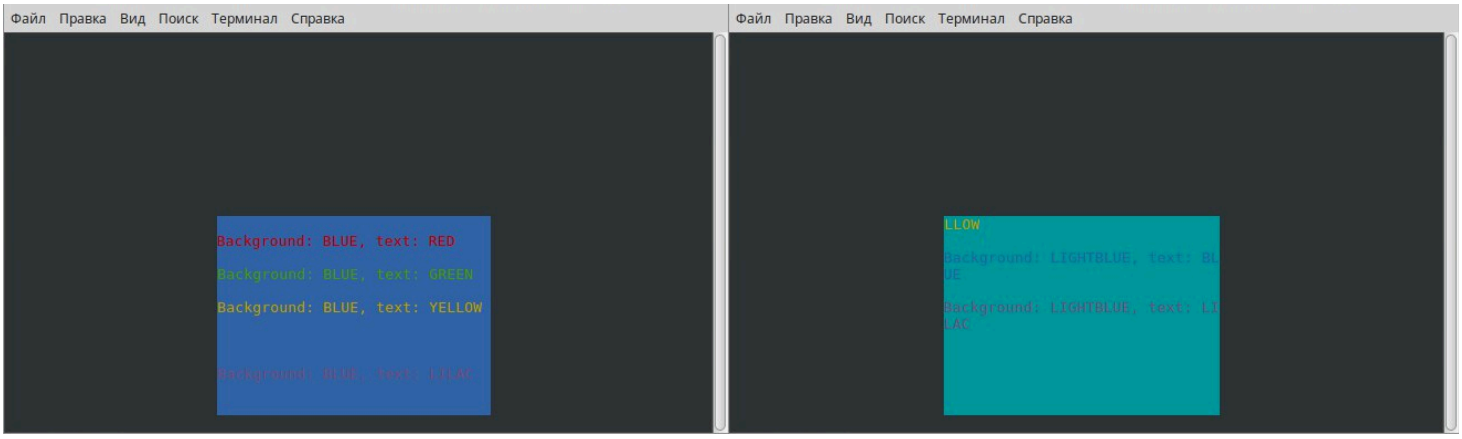
Функция textcolor(int newcolor) задает цвет символов, не затрагивая установленный цвет фона. Цвет может быть или числом, или формироваться из символических констант, значения которых определяет перечислимый тип COLORS.

Функция textbackground(int newcolor) задает цвет фона символов, не затрагивая установленный цвет символа. Цвет может быть или числом, или формироваться из символических констант, значения которых определяет перечислимый тип COLORS. Для цвета фона выбор ограничен значениями цветов 0-7. Если для цвета фона выбирается значение 8 - 15, то символы будут мерцать, так как бит мерцания установится в единицу, но цвет фона будет соответствовать значениям 0-7.

Задание

Написать программу так, чтобы в окно с координатами (25,12, 55, 23) с шагами 0,6 (секунд) и 1 (строк) выводилась надпись при всех возможных комбинациях цвета фона и цвета символов. Для каждой комбинации цветов в окне должны выводиться номера или символьные обозначения цветов фона и символов.

Примеры запуска



Текст программы

Main.cpp

```
#include <iostream>
#include "work.h"

int main() {
    lab2();
    return 0;
}
```

---

work.h

```
#ifndef LAB_2_WORK_H
#define LAB_2_WORK_H

#include <ncurses.h>
#include <panel.h>
#include <malloc.h>
#include <cstring>
```

```
#if defined(_WIN32) || defined(_WIN64)
#include <windows.h>
#define msleep(msec) Sleep(msec)
#else
#include <unistd.h>

#include "conio.h"

#define msleep(msec) usleep(msec*1000)
#endif

#include <string>

int lab2() {
    initscr();

    /******************/
    curs_set(0);
    char* colors[8] = {"BLACK", "RED", "GREEN", "YELLOW",
                                    "BLUE", "LILAC", "LIGHTBLUE", "WHITE"};
    textbackground(0);
    window(25, 12, 55, 23);
    std::string background_name, font_name;
    while (true) {
        for (int i = 0; i < 8; ++i) {
            textbackground(i);
            for (int j = 0; j < 8; ++j) {
                textcolor(j);
                cprintf("Background: %s, text: %s\n\n", colors[i], colors[j]);
                msleep(600);
            }
        }
    }

    /*****************/
    endwin();
    return 0;
}

#endif //LAB_2_WORK_H
```

---

conio.h

```
#ifndef __NCURSES_H
#include <ncurses.h>
#endif

#ifndef __CONIO_H
#define __CONIO_H
#endif

#define MAX_OKIEN 256

#define BLACK       0
#define RED         1
#define GREEN       2
#define BROWN       3
#define BLUE        4
#define MAGENTA     5
#define CYAN        6
```

```
#define LIGHTGRAY   7
#define DARKGRAY    8
#define LIGHTRED    9
#define LIGHTGREEN  10
#define YELLOW      11
#define LIGHTBLUE   12
#define PINK        13
#define LIGHTCYAN   14
#define WHITE       15

// ------------------------------- globalne ------------------

//int (* wsk_f)(void) = getch;

#undef getch
#define getch CURSgetch

#undef getche
#define getche CURSgetche


void inicjuj();

class Startuj   // konstruktor i destruktor klasy beda odpowiedzalni
{       public:     // za automagiczna inicjalizacje ustawien ;-)
        Startuj(){ inicjuj(); }
        ~Startuj(){ endwin(); }
} Start;                                // inicjuj!

typedef struct
{
        int       xup;
        int       yup;
        int       xdown;
        int       ydown;
        WINDOW*       okno;
} Okno;

bool    zainicjowane = FALSE; //czy juz po initscr() ?
int     znakSpecjalny = -1; //potrzebne do getch'a
int     n = 0; //liczba uzytych okienek

short   kolorTekstu = COLOR_WHITE;
short   kolorTla = COLOR_BLACK;
short   biezacaPara;

Okno    okienka[MAX_OKIEN];         //tablica struktur aktywnych okienek
WINDOW*     aktywneOkno = NULL;         //wsk na aktywne okno



// ----------------------------- koniec globalnych ------------

void inicjuj()
{
        initscr();
        start_color(); //wlaczmy kolorki
        cbreak(); //wylaczmy buforowanie wejscia
        noecho(); //bez wyswietlania na ekran
        //raw(); //nadpisywane i tak przez noecho
```

```
	keypad(stdscr, TRUE);
	scrollok(stdscr, TRUE);

	//domyslne okno
	aktywneOkno = stdscr;
	zainicjowane = TRUE;

	//utworzmy macierz 8x8 kolorow tla i tekstu
	short kolor = 1;
	for(short i=0; i<8; i++)
	{
		for(short j=0; j<8; j++, kolor++)
		{
			init_pair(kolor,i,j);
			if(i == COLOR_WHITE && j == COLOR_BLACK)
			//ustawmy czarne tlo i bialey tekst jako standard
			{
				biezacaPara = kolor;
			}
		}
	}

	wrefresh(aktywneOkno);
}

int simple_strlen(char* str)
{
	char* p;
	for(p = str; *p != 0; p++);
	return p-str;
}

void cputs(char* str)
{
	waddstr(aktywneOkno, str);
	wrefresh(aktywneOkno);
}

char* cgets(char* str)
{ // nie wiem dokladnie jak dziala orginalna f. cgets bo nie mam
  // do niej referencji..
	if(str == NULL || *str == 0)
	{
		*(str+1) = 0;
		return NULL;
	}

	int max = (int)(*str);

	echo();

	if(wgetnstr(aktywneOkno, (str + 2), max) == ERR)
	{
		*(str+1) = 0;
		return NULL;
	}

	noecho();

	*(str+1) = (char)simple_strlen(str+2);
```

```
        return str+2;
}

void clreol()
{
        wclrtoeol(aktywneOkno);
        wrefresh(aktywneOkno);
}

void clrscr()
{
        if(!zainicjowane) inicjuj();
        wbkgd(aktywneOkno, COLOR_PAIR(biezacaPara));
        //trzeba przesunac kursor? chyba nie...
        wclear(aktywneOkno);
}

int cprintf(char *fmt, ...)
// czysty hardcore ;-)
{
        if(!zainicjowane) inicjuj();

        va_list ap;
        va_start(ap, fmt);

      int i = vwprintw(aktywneOkno,fmt, ap); //jakie proste ;-)

        va_end(ap);

        wrefresh(aktywneOkno);

        return i;
}

int cscanf(char *fmt, ...)
{
        if(!zainicjowane) inicjuj();

        echo();

        va_list ap;
        va_start(ap, fmt);

        int i = vwscanw(aktywneOkno, fmt, ap);

        va_end(ap);

        wrefresh(aktywneOkno);
        noecho();

        return i;
}

int CURSgetch()
{
        if(!zainicjowane) inicjuj();

        int znak;
```

```
	if(znakSpecjalny>0) //drugi czlon znaku specjalnego 0x00 i 0x??
	{
		//zamieniamy znak na kod DOSowy - conio.h
		znak = znakSpecjalny;
		znakSpecjalny = -1;

		return znak-265+59;
	}

	znak = wgetch(aktywneOkno);

	if(znak > 255) //to mamy znak specjalny 0x00
	{
		znakSpecjalny = znak;
		return 0;
	}

	return znak;
}

int CURSgetche()
{
	echo();
	int znak = getch();
	noecho();
	return znak;
}

int gotoxy(int x, int y)
{
	if(!zainicjowane) inicjuj();
	wmove(aktywneOkno, y - 1, x - 1);
	return 0;
}

int kbhit()
{
	int znak;
	wtimeout(aktywneOkno, 0);
	znak = wgetch(aktywneOkno);
	//wtimeout(aktywneOkno, -1);
	nodelay(aktywneOkno, FALSE);
	if (znak == ERR) return 0;
	ungetch(znak);
	return 1;
}

int putch(int znak)
{
	wechochar(aktywneOkno,znak);
}

void textbackground(short kolor)
{
	if(!zainicjowane) inicjuj();
	kolorTla = kolor%8;
	short k=1;
	for(short i=0; i<8; i++) //wyszukajmy numer pary dla kolorow
	{
		for(short j=0; j<8; j++, k++)
```

```
                {
                        if(kolorTekstu == i && kolorTla == j)
                        {
                                biezacaPara = k;
                                wbkgd(aktywneOkno, COLOR_PAIR(k));
                        }
                }
        }

        wrefresh(aktywneOkno);
}

void textcolor(short kolor)
{
        if(!zainicjowane) inicjuj();
        kolorTekstu = kolor%8;

        short k=1;
        for(short i=0; i<8; i++) //wyszukajmy numer pary dla kolorow
        {
                for(short j=0; j<8; j++, k++)
                {
                        if(kolorTekstu == i && kolorTla == j)
                        {
                                biezacaPara = k;
                                wcolor_set(aktywneOkno,k, NULL);
                        }
                }
        }

        wrefresh(aktywneOkno);
}

int wherex(void)
{
        if(!zainicjowane) inicjuj();
        int x, y;
        getyx(aktywneOkno, y, x);
        return x + 1;
}

int wherey(void)
{
        if(!zainicjowane) inicjuj();
        int x, y;
        getyx(aktywneOkno, y, x);
        return y + 1;
}

void window(int xup, int yup, int xdown, int ydown)
{
        if( xup<1 || yup<1 || xdown>COLS || ydown>LINES)
        { //jesli zle dane podano...
                xdown = COLS - xup;
                ydown = LINES - yup;
                //return;
        }

        bool istnieje = FALSE;
```

```
	if(!zainicjowane) inicjuj();

	/*
	Istnieje alternatywne rozwiazanie tworzenia nowych okien,
	w momencie tworzenia nowego okna, usuwa sie okno poprzednie,
	tzn zwalnia pamiec tego okna, komenda delwin(nzw_okna) i tworzy
	sie nowe okno, ustawiajac jego jako domyslne-biezace. Jednak
	poniewaz moze to zabierac za duzo czasu i niepotrzebnie spowolniac,
	majac na uwadze rozmiar dzisiejszych pamieci, postanowilem, uzyc
	tablicy, ktora przechowywuje wsk. na adresy okien i wykorzystuje
	zaalokowana juz przestrzen. Aczkolwiek mozna to w kazdej chwili zmienic.
	*/

	for(int i=0; i<n && !istnieje; i++) //sprawdzimy czy podane okno juz nie
	// zostalo wczesniej stworzone
	{
		if( okienka[i].xup == xup && okienka[i].yup == yup
		&& okienka[i].xdown == xdown && okienka[i].ydown == ydown)
		{
			aktywneOkno = okienka[i].okno;
			istnieje = TRUE;
			clrscr();
		}
	}

	if(!istnieje && n < MAX_OKIEN) //jesli nie ma takiego okna to tworzymy je
	{
		aktywneOkno = newwin(ydown - yup + 1, xdown - xup + 1, yup - 1, xup - 1);
		//nie dam glowy czy dokladnie tak wyswietla conio.h

		//do tablicy zapisac...
		okienka[n].okno = aktywneOkno;
		okienka[n].xup = xup;
		okienka[n].yup = yup;
		okienka[n].xdown = xdown;
		okienka[n].ydown = ydown;

		wcolor_set(aktywneOkno,biezacaPara, NULL);
		wbkgd(aktywneOkno, COLOR_PAIR(biezacaPara));

		//przywrocenie ustawien klawiszy
		cbreak(); //wylaczmy buforowanie wejscia
		noecho(); //bez wyswietlania na ekran
		keypad(aktywneOkno, TRUE); //pelne kody klawiszy
		scrollok(aktywneOkno, TRUE);

		n++;
	}

	wrefresh(aktywneOkno);

	return;
}
```