**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра информационных систем**

отчет

**по лабораторной работе №2**

**по дисциплине «Организация ЭВМ и систем»**

**Тема: Исследование видеосистемы (текстовый режим).**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3376 |  | Михайлов Н.Ф. и Прошунина С.А. |
| Преподаватель |  | Ильин С.Е. |

Санкт-Петербург

2024

**Теоретические данные.**

**Общие положения**

Аппаратные средства для вывода информации на экран включают специальную электронную плату (видеоадаптер, либо адаптер дисплея, либо просто адаптер) и монитор (или просто экран). Конструктивно видеоадаптеры — это весьма сложные устройства, управляемые собственным микропроцессором, сравнимым по мощности с центральным процессором компьютера. Несмотря на огромное разнообразие фирм-производителей видеоадаптеров, имеется несколько стандартов, которым все эти продукты удовлетворяют.

В самом общем виде видеоадаптер состоит из двух основных частей: контроллера и видеопамяти (видеобуфера). Помимо этих обязательных узлов, наиболее совершенные видеоадаптеры имеют в своем составе ряд дополнительных узлов, например специализированные контроллеры быстрой манипуляции содержимым видеобуфера (так называемые контроллеры графики).

Память, необходимая для хранения полного образа экрана, называется видеостраницей.

Управление параметрами видеосистемы может выполняться на двух уровнях:

1. на уровне портов видеоадаптера;
2. обращением к функциям BIOS.

**Видеорежимы и их краткая 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.

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.

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.

Вывод на монитор содержимого видеобуфера происходит, начиная с неко­торого начального адреса, называемого смещением до видеостраницы. Страница 0 имеет нулевое смещение.

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.

**Функции консольного ввода-вывода**

Функции консольного ввода-вывода С++ помещены в файле <conio.h>.

Эти функции используют понятие активного окна экрана. Активное окно - это прямоугольная область экрана, в границах которой в данный момент работают функции. Описание активного окна (или, как часто говорят, фрейм) хранится во внутренней структурной переменной С++. Установку параметров активного текстового окна выполняет функция

window(int , int ,int , int );.

Она описывает активное текстовое окно: первая пара аргументов задает соответственно номера столбца и строки левого верхнего угла, вторая пара - правого нижнего угла. Строки и столбцы нумеруются, начиная от 1. Поэтому, например, координаты левого верхнего и правого нижнего углов экрана в режимах "25 строк х 80 столбцов" задаются парами (1,1) и (80,25). Расположение осей X и Y на экране показано на рис. 2.1.

Фрейм окна С++ имеет следующую структуру:

struct text\_info

{unsigned char

winright, winbottom; /\* столбец, строка правого нижнего угла \*/

attribute, normattr; /\* атрибуты окна\*/

currmode; /\* текущий режим работы видеоадаптера \*/

screenheight; /\* полная высота экрана \*/

screenwidth; /\* полная ширина экрана \*/

сurх, сurу; }; /\* строка, столбец текущей позиции курсора \*/

Информация об активном окне доступна при выполнении функции

gettextinfo( struct text\_info \*t);

При вызове эта функция заполняет поля структурной переменной, описанной по шаблону text\_info, указатель t на которую она получает.

**Управление курсором**

Видеоадаптеры всех типов аппаратно поддерживают курсор, который в текстовых режимах отображается на экране в виде одной или нескольких линий в пределах тексела. Курсор указывает на текущую позицию экрана (строку и столбец тексела), в которую будет записываться или из которой будет читаться средствами BIOS символ. При переключении адаптера в графический режим курсор становится невидимым, но BIOS сохраняет возможность изменять его позицию. Среди функций консольного ввода-вывода С++ текущей позицией курсора в окне управляет функция gotoxy(int x, int y);.

Устанавливает курсор в заданную строку y и столбец x в текущем активном окне экрана. Верхний левый угол окна имеет координаты (1,1). При попытке по­зиционировать курсор за границы окна он останавливается на границе окна. Особенностью функции является то, что координаты задаются относительно левого верхнего угла текущего окна.

Текущую позицию x и y курсора в активном текстовом окне можно узнать при вызове соответственно функций wherex () и wherey().

Эти функции соответственно возвращают номер столбца и номер строки текущей позиции курсора. Кроме того, текущая позиция курсора в окне возвращается в полях curx и сurу структурной переменной, заполняемой при вызове функции gettextinfo( ).

**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.

## Скроллинг. Очистка окна и всего экрана

Функции АН = 06 и 07 прерывания 10h BIOS осуществляют так называемый скроллинг (прокрутку) окна экрана. пример Си-функции, выполняющей вертикальный скроллинг окна экрана, заданного строкой и столбцом левого верхнего (l\_row, l\_col) и строкой и столбцом правого нижнего (r\_row, r\_col) углов окна. Если переменная direction равна UP, происходит скроллинг на одну строку вверх, если она равна DOWN - скроллинг на одну строку вниз, если ENTIRE -выполняется очистка окна. Добавляется строка с атрибутом attr.

#include <CONIO.H>

#include < dos. h >

enum { ENTIRE, UP, DOWN };

void scroll (int direction, char l\_row, char l\_col, char r\_row, char r\_col, char attr)

{

union REGS r;

if (direction == DOWN) r.h.ah = 7;

else r.h.ah = 6;

r.h.al = lines; // шаг скроллинга

r.h.ch = l\_row; r.h.cl= l\_col; r.h.dh = r\_row; r.h.dl= r\_col;

r.h.bh=attr;

int86(0x10,&r,&r);

}

Если окно занимает весь экран и задается direction = ENTIRE, происходит фактическая очистка всего экрана и его "заливка" цветом, задаваемым атрибутом чистой строки attr.

**Вывод информации в окно экрана**

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.

Аналог clrscr() – использование system("cls"); (добавить #include <stdlib.h>)

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.

Функция cputs( char \*str) выводит строку символов в текстовое окно, начиная с текущей позиции курсора.

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.

Функция putch(int ch) выводит символ в текущей позиции текстового окна экрана.

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.

Функции highvideo (void), lowvideo (void) и normvideo (void) задают соответственно использование повышенной, пониженной и нормальной яркости для последующего вывода символов на экран.

Функция textattr(int newattr) устанавливает атрибут для функций, работающих с текстовыми окнами.

Функция textcolor(int newcolor) задает цвет символов, не затрагивая установленный цвет фона.

Функция textbackground(int newcolor) задает цвет фона символов, не затрагивая установленный цвет символа.

**Задание на лабораторную работу.**

1. Написать программу, в которой в окно с координатами (x1 = 25, у1 = 10, х2 = 55, у2 = 20) с шагом 0,2 секунд и 3 строки, выводится строка при всех возможных комбинациях цвета фона и цвета символов. Строка содержит обозначение цвета фона и символа. Для каждой комбинации цветов в окне должны выводиться номера цветов фона и символов. Цвет окна должен соответствовать цвету фона.

2. Дополнить программу скроллингом окна с направлением: DOWN, используя функции прерывания 10h BIOS.

**Текст программы**

**1 задание:**

#include <conio.h>

#include <dos.h>

#include <bios.h>

int main() {

window(25, 10, 55, 20);

for (int i = 0; i < 16\*16; i++) {

textcolor(i % 16);

if (i % 16 == 0) textbackground(15 - ((i / 16) % 16));

cprintf("Text:%d Background:%d\n\n\n\r", i % 16,15 - ((i / 16) % 16));

delay(200);

}

return 0;

}

**2 задание:**

include <conio.h>

#include <dos.h>

enum ScrollDirection { ENTIRE = 0, UP = 1, DOWN = 2 };

void scroll (int direction, char l\_row, char l\_col, char r\_row, char r\_col, char attr, int lines)

{

union REGS r;

if (direction == DOWN) {

r.h.ah = 7;

}

else r.h.ah = 6;

r.h.al = lines;

r.h.ch = l\_row;

r.h.cl= l\_col;

r.h.dh = r\_row;

r.h.dl= r\_col;

r.h.bh=attr;

int86(0x10,&r,&r);

}

int main() {

window(25, 10, 55, 20);

clrscr();

for (int i = 0; i < 16\*16 ;i++) {

int text = i % 16;

int background = 15 - ((i / 16) % 16);

textcolor(text);

textbackground(background);

int scrollBack = 16 \* background + text;

scroll(DOWN, 9, 24, 19, 54, scrollBack, 3);

cprintf("Text:%d Background:%d\r", text, background);

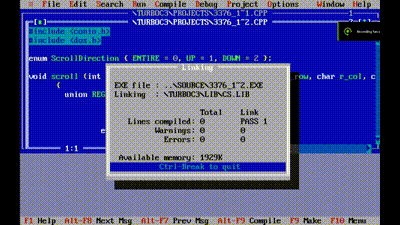
delay(200);

}

return 0;

}

**Примеры запуска и работы программы**

*Рис.1 Пример запуска первой программы*

*Рис.2 Пример запуска второй программы*