МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

“ЛЭТИ” ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра ВТ

ОТЧЕТ

по лабораторной работе №2

по дисциплине “ОЭВМиС”

Тема: ИССЛЕДОВАНИЕ ВИДЕОСИСТЕМЫ

(ТЕКСТОВЫЙ РЕЖИМ)

Студент гр. 4363 Карачев П.А.

Преподаватель Павлов С.М.

Санкт-Петербург

2016

**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). Основное назначение видеобуфера - хранение образа информации экрана. Видеоадаптер 25 и более раз в секунду формирует изображение на экране. Так как человеческий глаз не способен уловить такое быстрое мелькание кадров, создается иллюзия неподвижного изображения на экране монитора. Изображение на экране строится из небольших точек - так называемых пикселов (pixel - Picture ELement). Число пикселов в строке и число самих строк различно для разных типов видеоадаптеров.

Память, необходимая для хранения полного образа экрана, называется видеостраницей. Часто общий объем видеопамяти намного превышает объем страницы. В этом случае появляется возможность хранить в видеобуфере не одну, а несколько страниц. Та видеостраница, которая постоянно "освежается" в данный момент, называется текущей. Видеоадаптер способен выполнять переключение текущей видеостраницы. Объем видеопамяти и число возможных страниц, зависит от конкретного адаптера.

Управление параметрами видеосистемы может выполняться на двух уровнях:

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

В данной части описывается управление лишь некоторыми из параметров видеосистемы: определение типа и характеристик видеоадаптера и монитора, задание формы курсора и его позиции на экране, выбор режима, видеостраницы и палитры.

## ***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. 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 | Макси-  мальное  число  страниц | Разрешение | | Адрес  видео-  буфера | 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).

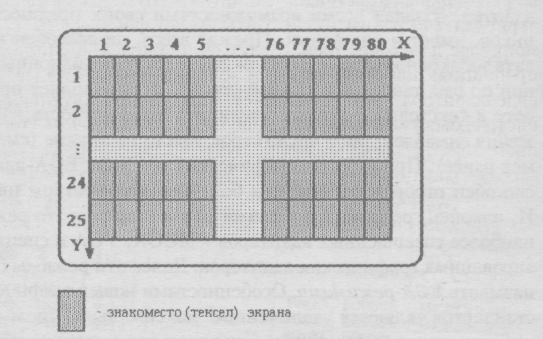


Рис. 2.1. Представление экрана в текстовых режимах "25 строк х 80 столбцов"

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, В -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 х 25 строк", то для хранения полного образа экрана (видеостраницы) требуется 25 х 40 х 2 = 2000 байт видеопамяти. В режимах "80 столбцов х 25 строк" видеостраница занимает уже 25 х 80 х 2 = 4000 байт. Минимальная конфигурация видеоадаптера CGA имеет обычно 16К 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 х 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).

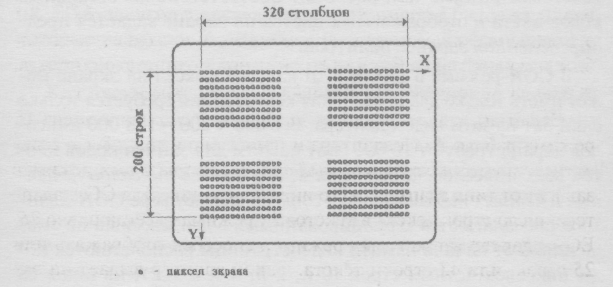


Рис. 2.3. Представление экрана в графических режимах 4,5 (320 столбцов х 200 строк)

## ***Функции консольного ввода-вывода***

Функции консольного ввода-вывода С++ помещены в файле <conio.h>, предназначены для облегчения работы по созданию простейшего оконного интерфейса. 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.

Функция window() инициализирует поля координат фрейма окна. Функции textcolor(), textbackground(), textattr() и другие управляют цветом отображаемых символов в окне. Они будут рассмотрены далее.

## ***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. Специальные регистры видеоконтроллера хранят текущую позицию и форму курсора.

Программное прерывание 10h BIOS имеет в своем составе специальные функции для установки формы курсора, чтения и установки его координат.

Функция АН = 0lh задает высоту курсора. Регистр СН определяет номер верхней телевизионной линии, a CL - номер нижней линии при изображении курсора. Например, задав значения СН = 0, CL = 13, получим курсор, занимающий все знакоместо. Значения СН = 6, CL = 7 устанавливают форму курсора по умолчанию - две равномерно мерцающие строки в нижней части тексела. Если СН > CL, курсор будет состоять из двух групп линий вверху и внизу знакоместа с разрывом посередине. BIOS поддерживает одинаковую форму курсора для всех видеостраниц текста. Биты 5 и 6 кода СН управляют мерцанием и могут выключить ото­бражение курсора (табл. 2.2).

Табл. 2.2. Биты управления отображением курсора

|  |  |  |
| --- | --- | --- |
| Биты регистра СН | | Действие, оказываемое на курсор |
| бит 6 | бит 5 |
| 0 | 0 | Видимый, мерцающий равномерно с нормальной скоростью курсор |
| 0 | 1 | Курсор не отображается |
| 1 | 0 | Видимый, мерцающий равномерно с повышенной скоростью курсор |
| 1 | 1 | Видимый, неравномерно мерцающий курсор |

BIOS записывает текущую форму курсора в слово по адресу 0040:0060h, при этом младший байт содержит номер нижней строки, а старший - номер верхней строки для отображения курсора.

Управление формой курсора находит ограниченное применение в практике программирования. Однако изменением формы курсора можно отображать различные режимы работы программы, например режим "Вставка" или режим "Замена". Намного чаще приходится выключать курсор. Это необходимо, когда программа сама отображает курсор.

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.

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 С могут по выбору пользователя работать и на самом нижнем уровне, выполняя доступ к видеобуферу при работе в текстовом режиме.

## ***Скроллинг. Очистка окна и всего экрана***

Функции АН = 06 и 07 прерывания 10h BIOS осуществляют так называемый скроллинг (прокрутку) окна экрана. При выполнении скроллинга на одну строку вверх вся информа­ция в окне перемещается на строку вверх. Внизу окна появляется чистая строка. При выполнении скроллинга на одну строку вниз содержимое окна сдвигается на строку вниз и вверху окна добавляется чистая строка. Значение регистра AL задает число строк, на которое выполняется скроллинг. Если AL=0, выполняется очистка окна. Значения в СН и CL определяют строку и столбец левого верхнего угла окна, в DH и DL -строку и столбец правого нижнего угла. Строки и столбцы нумеруются от 0. Значение в регистре ВН задает атрибут добавляемой чистой строки.

Приведем 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 < dos. h >

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

{

union REGS r;

if (direction)

{ r. h. a= 1; r.h. ah=direction; }

else

{r.h.al=0; r.h.ah=6; }

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. Например, для очистки экрана в режимах с 25 строками и 80 столбцами вызов функции может иметь вид:

scroll(ENTIRE,0,0,24, 79,0x07);

Для получения цветной рамки по периметру всего экрана можно выполнить два обращения:

scroll (ENTIRE, 0,0, 24, 79, color);

scroll(ENTIRE, 1, 1, 23, 78, 0x07);

Здесь значение color - атрибут. Цвет рамки будет совпадать с цветом фона символа, так как символом является пробел.

Скроллинг окна средствами BIOS возможен как в текстовых, так и в графических режимах работы видеоадаптера. Скроллинг в графических режимах разных адаптеров может иметь некоторые особенности.

Одним из интересных применений скроллинга является построение "взрывающихся" окон (exploding windows). Такие окна "вырастают" на экране из определенного места (середины, одного из углов, сначала заполняется средняя горизонталь, а затем окно раздвигается вверх и т.п.). Секрет построения "взрывающихся" окон прост. Начиная с определенного места окна (например, его центра), выполняется очистка первого, самого маленького окна. После этого по периметру этого окна рисуется рамка. Затем координаты левого верхнего и правого нижнего углов модифицируются, и все повторяется: очистка старой рамки скроллингом, вывод новой, большего размера, модификация координат углов и так далее до тех пор, пока окно не "вырастет" до нормальных размеров.

## ***Вывод информации в окно экрана***

С++ 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.

**Вариант 3**

**Цель работы:** изучение работы с видеосистемой в текстовом режиме, освоение приемов использования цветовой палитры: измене­ние цвета символов и фона на всем экране и в отдельном окне.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Алгоритм выполняемого задания.**

Шаг 1.

Выбираем цвет консоли:

// 0 = Черный 8 = Серый

// 1 = Синий 9 = Светло-синий

// 2 = Зеленый A = Светло-зеленый

// 3 = Голубой B = Светло-голубой

// 4 = Красный C = Светло-красный

// 5 = Лиловый D = Светло-лиловый

// 6 = Желтый E = Светло-желтый

// 7 = Белый F = Ярко-белый

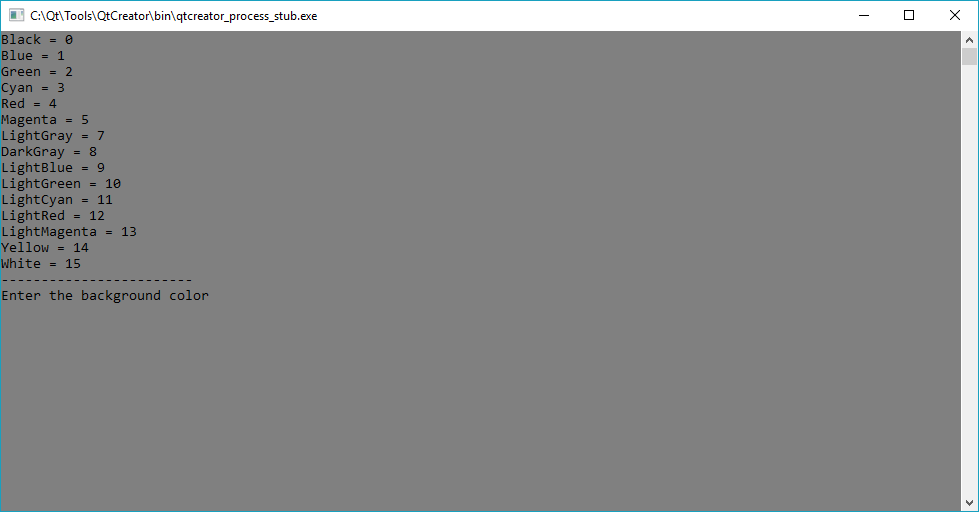
system("color 80");

Шаг 2.

Выбираем цвет выделения текста:

printf("Enter the background color ");

scanf("%u", &numBg);

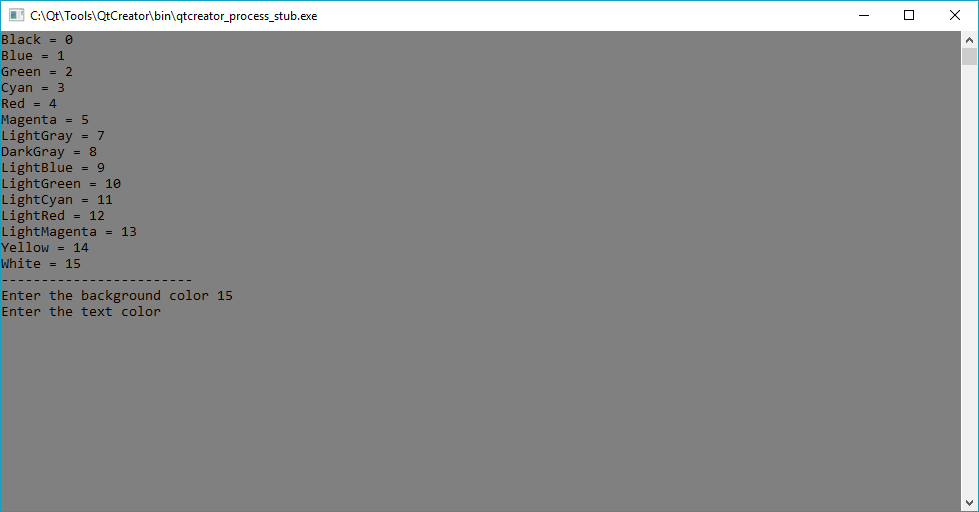


Шаг 3.

Выбираем цвет текста:

printf("Enter the text color ");

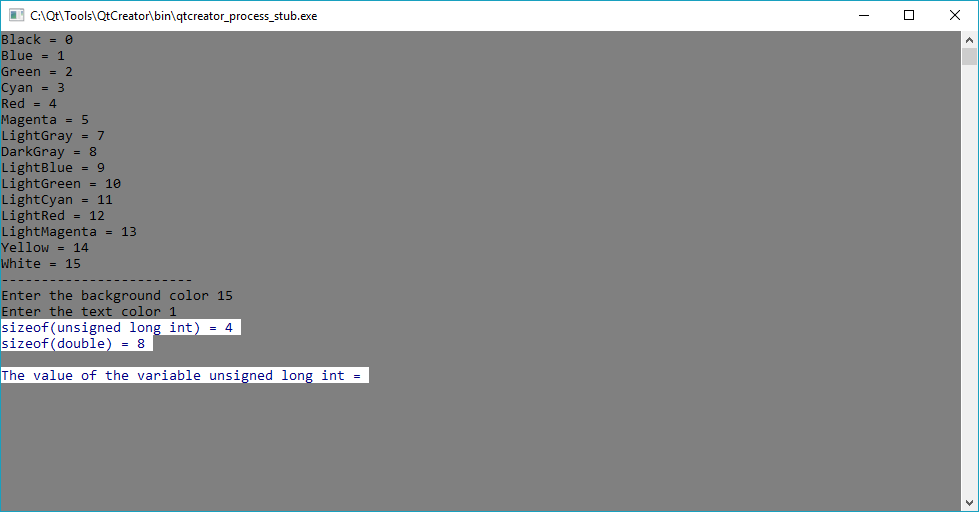
scanf("%u", &numCol);



Шаг 4.

Операция по изменению цвета выделения и цвета текста.

SetConsoleTextAttribute(hConsole, (WORD) ((numBg << 4) | numCol));



**Пример кода:**

enum ConsoleColor {

Black = 0,

Blue = 1,

Green = 2,

Cyan = 3,

Red = 4,

Magenta = 5,

Brown = 6,

LightGray = 7,

DarkGray = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15

};

int main(void)

{

int numBg;

int numCol;

/\*Получение дескриптора\*/

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

// Атрибуты цветов задаются в виде ДВУХ шестнадцатеричных цифр — первая

// задает цвет фона, а вторая определяет цвет переднего плана. Каждая цифра

// может иметь следующие значения:

// 0 = Черный 8 = Серый

// 1 = Синий 9 = Светло-синий

// 2 = Зеленый A = Светло-зеленый

// 3 = Голубой B = Светло-голубой

// 4 = Красный C = Светло-красный

// 5 = Лиловый D = Светло-лиловый

// 6 = Желтый E = Светло-желтый

// 7 = Белый F = Ярко-белый

system("color 80");

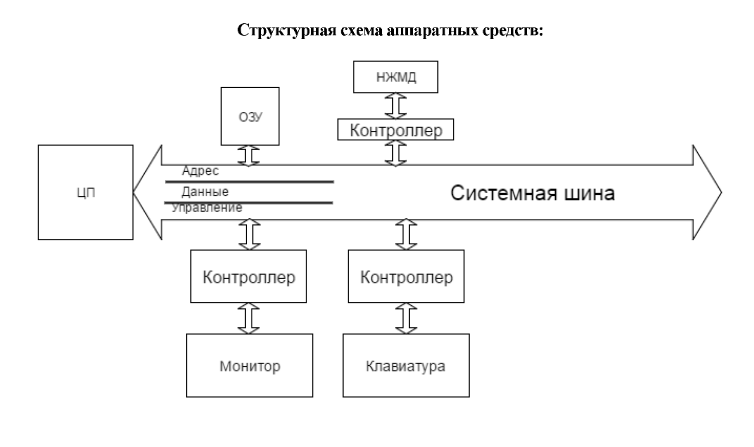
printf("Enter the background color ");

scanf("%u", &numBg);

printf("Enter the text color ");

scanf("%u", &numCol);

SetConsoleTextAttribute(hConsole, (WORD) ((numBg << 4) | numCol));



АЛУ-арифметико-логическое устройство

УУ-устройство управления

ОЗУ-оперативное запоминающее устройство

ПЗУ-постоянное запоминающее устройство

**Вывод**

Было изучена работа с видеосистемой в текстовом режиме, освоение приемов использования цветовой палитры: измене­ние цвета символов и фона на всем экране и в отдельном окне на машине c ЦП Intel® Core™ i3-6100U CPU @ 2.30GHz построенного на архитектуре x64\_32 в OC Windows 10.