**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №2**

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

Тема: Исследование видеосистемы (текстовый режим)

|  |  |  |
| --- | --- | --- |
| Студент гр. 1376 |  | Прохоренко К.Д. |
| Преподаватель |  | Ельчанинов М.Н. |

Санкт-Петербург

2022

**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.

**Задание на лабораторную работу.**

1. Изменить программу, полученную на предыдущей работе таким образом, чтобы в окно с координатами (20, 15, 60, 20) с шагами Т (0,9) и строк S (1) 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.
2. Организовать в окне вывод разноцветных сообщений со скроллингом окна.

**Краткие сведения о видеосистемах ПЭВМ, текстовом режиме их работы и функциях обслуживания текстового режима.**

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.

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). 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.

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.

**Алгоритмы**

Для получения двоичного представления числа типа long double (64) была использована конструкция union:

**union** longdouble

{

long double ld;

\_\_int128\_t ldint;

};

что представляет собой объединение типов данных unsigned long long и \_\_int128\_t в памяти (для двух переменных ld и ldint, (см. рис. 1), которое дает возможность записывать вещественное число в переменную типа long double и читать его как целый тип (\_\_int128\_t).

У обеих типов используется одно и тоже место в памяти, т.е. если записать в ld значение с плавающей точкой, то можно его взять как **\_\_int128\_t**и сделать побитовые операции.

Для определения окна с координатами (x1, у1, х2, у2) используется функция setWindow, которая реализована на средствах windowsAPI.

Для задержки вывода (шаг Т (секунд)) была использована функция sleep(), которая в качестве аргумента принимает значение времени в миллисекундах. Для шага S (строк) используется функция scroll(), которая реализована на средствах windowsAPI.

Для вывода надписи при всех возможных комбинациях цвета фона и цвета символов было использовано две конструкции switch-case (первая для фона, вторая для текста) и цикл do while.

**Текст программы**

#include <iostream>

#include <conio.h>

#include <string>

#include <vector>

#include <windows.h>

**using** **namespace** std;

**union** longdouble

{

long double ld;

\_\_int128\_t ldint;

};

**inline** void clearbuffer()

{

**while**(kbhit()) getch();

}

void tobinary(char data)

{

unsigned char mask;

**for** (mask = 128; mask != 0; mask >>= 1)

{

bool a = (data & mask);

std::cout << a;

}

std::cout << " ";

}

**template** <**class** **typeOfData**>

void output(typeOfData data)

{

short size = 0;

**if**(**sizeof**(data) == 16) size = 10;

**if**(**sizeof**(data) == 2) size = 2;

**for** (short i = size; i != 0; i--)

{

tobinary(data >> (8 \* (i - 1)));

}

}

bool range(char s)

{

**if** ((s >= '0' && s <= '9') ) **return** 1;

**else** **return** 0;

}

long double input(long double data)

{

char s;

string result;

**do**

{

clearbuffer();

s = \_getch();

**if** (range(s))

{

std::cout << char(s);

result.push\_back(s);

}

**else** **if** (s == 8)

{

**if** (!result.empty())

{

result.pop\_back();

std::cout << "**\b** **\b**";

}

}

**else** **if** (s == '-' && result.empty())

{

std::cout << char(s);

result.push\_back(s);

}

**else** **if** (s == '.')

{

std::cout << char(s);

result.push\_back(s);

}

} **while** (s != 13);

std::cout << endl;

long double ans = 0;

long double ans\_f = 0;

short sign = 1;

**if** (result[0] == '-')

{

sign = -1;

result.erase(0,1);

}

**for**(int i = 0; i < result.size(); i++)

{

**if** (result[i] == 0) result.erase(0,1);

**else** **break**;

}

**for**(int i = 0; i < result.size(); i++)

{

**if**(result[i] == '.') **break**;

ans = ans \* 10 + result[i] - '0';

}

**while** (result.back() != '.')

{

ans\_f = (ans\_f + (result.back())- '0') / 10;

result.pop\_back();

}

ans = sign \* (ans + ans\_f);

**return** ans;

}

short input(short data)

{

char s;

string result;

**do**

{

**while**(kbhit()) getch();

s = \_getch();

**if** (range(s))

{

std::cout << char(s);

result.push\_back(s);

}

**else** **if** (s == 8)

{

**if** (!result.empty())

{

result.pop\_back();

std::cout << "**\b** **\b**";

}

}

**else** **if** (s == '-' && result.empty())

{

std::cout << char(s);

result.push\_back(s);

}

} **while** (s != 13);

std::cout << endl;

short ans = 0;

short sign = 1;

**if** (result[0] == '-')

{

sign = -1;

result.erase(0,1);

}

**for**(int i = 0; i < result.size(); i++)

{

**if** (result[i] == 0) result.erase(0,1);

**else** **break**;

}

**for**(int i = 0; i < result.size(); i++)

{

ans = ans \* 10 + result[i] - '0';

}

**return** ans\*sign;

}

**template** <**class** **typeOfData**>

typeOfData swap\_bits(typeOfData data, int i, int j)

{

bool biti = data & ((typeOfData)1 << i);

bool bitj = data & ((typeOfData)1 << j);

**if** (biti != bitj)

{

typeOfData bit\_mask = ((typeOfData)1 << i) | ((typeOfData)1 << j);

data ^= bit\_mask;

**return** data;

}

**return** data;

}

**template** <**class** **typeOfData**>

typeOfData reversepairs(typeOfData data)

{

char s;

string result;

**do**

{

**while**(kbhit()) getch();

s = \_getch();

**if** (s >= '0' && s <= '9')

{

std::cout << s;

result.push\_back(s);

}

**else** **if** (s == 8)

{

**if** (!result.empty())

{

result.pop\_back();

std::cout << "**\b** **\b**";

}

}

**else** **if** (s == 32)

{

**if**(!result.empty())

{

result.push\_back(s);

std::cout << " ";

}

}

} **while** (s != 13);

vector <int> vresult;

int value = 0;

**for**(int i = 0; i < result.size(); i++)

{

**if**(result[i] != char(32))

{

value = value \* 10 + (result[i] - '0');

}

**if**(result[i] == char(32) || i == result.size() - 1)

{

vresult.push\_back(value);

value = 0;

}

}

**for**(int i = 0; i < result.size(); i+=2)

{

data = swap\_bits(data, vresult[i], vresult[i+1]);

}

**return** data;

}

void setWindow(short x1, short y1, short x2, short y2)

{

short width = x2 - x1;

short height = y2 - y1;

COORD bufferSize = {width, height};

SMALL\_RECT windowRS = {0, 0, 1, 1};

SMALL\_RECT windowSize = {0, 0, width - 1, height - 1};

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

SetConsoleWindowInfo(hStdOut, TRUE, &windowRS);

SetConsoleScreenBufferSize(hStdOut, bufferSize);

SetConsoleWindowInfo(hStdOut, TRUE, &windowSize);

}

**inline** void resetWindow(CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo)

{

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

SetConsoleScreenBufferSize(hStdOut, consoleInfo.dwSize);

SetConsoleWindowInfo(hStdOut, TRUE, &consoleInfo.srWindow);

SetConsoleTextAttribute(hStdOut, consoleInfo.wAttributes);

}

void scroll(int x = 1)

{

CONSOLE\_SCREEN\_BUFFER\_INFO csbiInfo;

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

GetConsoleScreenBufferInfo(h, &csbiInfo);

SMALL\_RECT srctScrollRect, srctClipRect;

CHAR\_INFO chiFill;

COORD coordDest;

srctScrollRect.Left = 0;

srctScrollRect.Top = 1;

srctScrollRect.Right = csbiInfo.dwSize.X - x;

srctScrollRect.Bottom = csbiInfo.dwSize.Y - x;

coordDest.X = 0;

coordDest.Y = 0;

srctClipRect = srctScrollRect;

srctClipRect.Top = 0;

chiFill.Attributes = FOREGROUND\_RED|FOREGROUND\_INTENSITY;

chiFill.Char.AsciiChar = ' ';

ScrollConsoleScreenBuffer

(

h,

&srctScrollRect,

&srctClipRect,

coordDest,

&chiFill

);

}

void printColor(char color)

{

unsigned char bg = color >> 4;

unsigned char text = color & 15;

std::cout << (int)bg << ' ';

**switch**(text)

{

**case** 0: std::cout << "BLACK "; **break**;

**case** 1: std::cout << "BLUE "; **break**;

**case** 2: std::cout << "GREEN "; **break**;

**case** 3: std::cout << "CYAN "; **break**;

**case** 4: std::cout << "RED "; **break**;

**case** 5: std::cout << "MAGENTA "; **break**;

**case** 6: std::cout << "BROWN "; **break**;

**case** 7: std::cout << "LIGHTGRAY "; **break**;

**case** 8: std::cout << "DARKGRAY "; **break**;

**case** 9: std::cout << "LIGHTBLUE "; **break**;

**case** 10: std::cout << "LIGHTGREEN "; **break**;

**case** 11: std::cout << "LIGHTCYAN "; **break**;

**case** 12: std::cout << "LIGHTRED "; **break**;

**case** 13: std::cout << "LIGHTMAGENTA "; **break**;

**case** 14: std::cout << "YELLOW "; **break**;

**case** 15: std::cout << "WHITE "; **break**;

}

}

**template** <**class** **typeOfData**>

void colorizedOutput(typeOfData changedData)

{

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

CONSOLE\_SCREEN\_BUFFER\_INFO startConsoleInfo;

GetConsoleScreenBufferInfo(hStdOut, &startConsoleInfo);

**for**(int i = 0; i < 128; i++)

{

scroll();

SetConsoleTextAttribute(hStdOut, i);

printColor(i);

output(changedData);

Sleep(900);

COORD cursor = {0, 4};

SetConsoleCursorPosition(hStdOut, cursor);

**if**(\_kbhit()) {clearbuffer(); **break**;}

}

}

**template** <**class** **typeOfData**>

void lab2(typeOfData changedData)

{

setlocale(LC\_ALL, "Russian");

std::cout << "lab2? Y/N";

char choice = \_getch();

**if**(choice == 'Y' || choice == 'y')

{

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

CONSOLE\_SCREEN\_BUFFER\_INFO consoleInfo;

GetConsoleScreenBufferInfo(hStdOut, &consoleInfo);

setWindow(20, 15, 60, 20);

colorizedOutput(changedData);

resetWindow(consoleInfo);

}

}

int main()

{

int menu;

**do**

{

system("cls");

std::cout << "1. short**\n**";

std::cout << "2. long double**\n**";

std::cout << "0. exit**\n**";

menu = \_getch();

**if**(menu == '1')

{

short data = 0;

std::cout << "input short int number: ";

data = input(data);

std::cout << "decimal: " << data << std::endl;

std::cout << "binary: ";

output(data);

std::cout << std::endl;

std::cout << "enter the number of bit pairs to be changed: ";

short reversedBits = reversepairs(data);

std::cout << "**\n\n**decimal: " << reversedBits << std::endl;

std::cout << "binary: ";

output(reversedBits); std::cout << std::endl << std::endl;

lab2(reversedBits);

}

**if**(menu == '2')

{

longdouble data;

std::cout << "**\n**input float number: ";

data.ld = input(data.ld);

std::cout << "decimal: " << data.ld << endl;

std::cout << "binary: ";

output(data.ldint);

std::cout << endl;

std::cout << "enter the number of bit pairs to be changed: ";

longdouble reversedBits;

reversedBits.ldint = reversepairs(data.ldint);

std::cout << "**\n\n**decimal: " << reversedBits.ld << std::endl;

std::cout << "binary: ";

output(reversedBits.ldint); std::cout << std::endl << std::endl;

lab2(reversedBits.ldint);

}

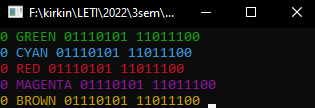
}

**while** (menu != '0');

}

**Примеры запуска программы**

На рисунках демонстрирована работа программы. Программа выводит на экран внутреннее представление чисел в двоичной системе счисления.



Монитор

