

Laboratory lesson №2

(The spring session)

Theoretical information

BMP (Bitmap Picture) is a bitmap storage format.

BMP works with a large number of applications, as its support is integrated into Windows and OS/2 operating systems. **BMP** files can have the extension .bmp, .dib and .rle. In addition, data in this format is included in RES resource binaries and PE files.

Color depth in this format can be 1, 2, 4, 8, 16, 24, 32, 48 bits per pixel, maximum image sizes are 65535×65535 pixels. However, the 2 bit depth is not officially supported.

The **BMP** format supports RLE compression, but now there are more compression formats, and because of the large size, **BMP** is rarely used on the Internet, where PNG and GIF are used for lossless compression.

The **BMP** file consists of four parts:

1. File title (**BITMAPFILEHEADER**)
2. Image title (**BITMAPINFOHEADER** may be missing).
3. The palette (may be missing).
4. Image.

The **BITMAPFILEHEADER** structure contains information about the type, size and representation of data in a file:

```
struct BitmapFileHeader {
    WORD    bfType; // file type, «BM» (0x424d),
                // type WORD - unsigned short
                // type DWORD - unsigned long
    DWORD    bfSize; // size of file in bytes.
    WORD     bfReserved1; // reserved, must be zero
    WORD     bfReserved2; // reserved, must be zero
    DWORD    bfOffBits; // The offset, in bytes, from the beginning of
//the BITMAPFILEHEADER structure to the bitmap bits.
};
```

The simplest version of the image title **BITMAPINFOHEADER**:

```
struct BitmapInfoHeader {
    DWORD    biSize; // size of structure in bytes
    LONG     biWidth; // the width of image, in pixels.
    LONG     biHeight; // the height of image, in pixels.
    WORD     biPlanes; // must be set to 1
    WORD     biBitCount; // Specifies the number of bits per pixel
    DWORD    biCompression; //the type of compression
// (0 - no compression)
    DWORD    biSizeImage; // the size, in bytes, of the image.
    LONG     biXPelsPerMeter; // the horizontal resolution, in pixels
//per meter, of the target device for the bitmap
// type LONG - long int
    LONG     biYPelsPerMeter; // the vertical resolution, in pixels
//per meter, of the target device for the bitmap
    DWORD    biClrUsed; //the number of color indices in the color
//table that are actually used by the bitmap
    DWORD    biClrImportant; // the number of color indices that are
//considered important for displaying the bitmap
};
```

The colors in the **.bmp** file are set in **RGB (Red-Green-Blue)** format. The **RGB** format is a set of rules by which any color can be represented as a specific code of numbers and letters.

The **RGB** format instructs the computer how to mix red, green and blue in different combinations to get all the colors of the rainbow (fig. 1). Each color: red, green or blue, characterized by its intensity or saturation.

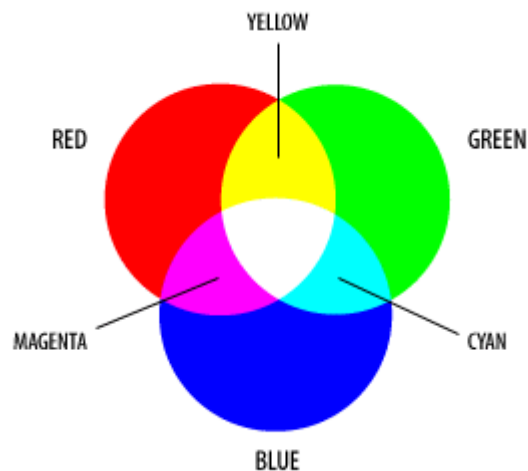


Рисунок 1.

The number of each color can range from 0 to 255.

Absolutely red will have the form of a record (255,0,0). This means that the amount of red 255, green 0 (no green component), blue 0 (no blue component).

Absolutely blue (0,255,0) and green (0,0,255).

With different combinations, different colors of the rainbow are already forming:

bright purple - (255,0,255), black - (0,0,0)

This is a record (255,0,255) in decimal form. But the RGB color can also be represented in hexadecimal number system. It is easier for a computer to operate such numbers.

If we convert each of the numbers corresponding to a particular color into a hexadecimal number system, we will get another form of color notation.

FFFFFF - (255,255,255) – white color.

FF is the number 255 in the hexadecimal number system.

000000 - (0,0,0) – black color.

RGB color can be represented in both hexadecimal and decimal numbers (fig. 2).

Indicates that it is the hexadecimal number system.

In C ++, the representation of hexadecimal numbers # is replaced by **0x**

































Named	Numeric	Color name	Hex rgb	Decimal
		<i>black</i>	#000000	0,0,0
		<i>silver</i>	#C0C0C0	192,192,192
		<i>gray</i>	#808080	128,128,128
		<i>white</i>	#FFFFFF	255,255,255
		<i>maroon</i>	#800000	128,0,0
		<i>red</i>	#FF0000	255,0,0
		<i>purple</i>	#800080	128,0,128
		<i>fuchsia</i>	#FF00FF	255,0,255
		<i>green</i>	#008000	0,128,0
		<i>lime</i>	#00FF00	0,255,0
		<i>olive</i>	#808000	128,128,0
		<i>yellow</i>	#FFFF00	255,255,0
		<i>navy</i>	#000080	0,0,128
		<i>blue</i>	#0000FF	0,0,255
		<i>teal</i>	#008080	0,128,128
		<i>aqua</i>	#00FFFF	0,255,255

Figure 2. Some standard colors that your computer can operate.

Example. Write a console application in which the user can set the width and height of the image; fill the image with red color and save the image as a .bmp file (Fig. 3).

```
#include <fstream>
#include <iostream>

using namespace std;

typedef uint16_t WORD;
typedef uint32_t DWORD;
typedef int32_t LONG;
//to avoid rounding the size of structure to the larger value of
//bits
#pragma pack(push,1)
struct __attribute__((__packed__)) BitmapFileHeader {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
};
#pragma pack(push,1)
struct __attribute__((__packed__)) BitmapInfoHeader {
    DWORD   biSize;
    LONG    biWidth;
    LONG    biHeight;
    WORD    biPlanes;
    WORD    biBitCount;
    DWORD   biCompression;
```

```

    DWORD biSizeImage;
    LONG biXPelsPerMeter;
    LONG biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
};

//the creation of .bmp file
// the setting parameters for the .bmp header file and image //fields
void CreateBmp(const char *fileName, unsigned int **colors , int
height, int width)
{
    BitmapFileHeader bfh = {0};
    BitmapInfoHeader bih = {0};

    bfh.bfType = 0x4D42; // the 'BM' symbols
    bfh.bfOffBits = sizeof(bfh) + sizeof(bih); //
    bfh.bfSize =bfh.bfOffBits + sizeof(colors[0][0])*width*height;
    // the final size of file

    bih.biSize = sizeof(bih); // the size of sructure
    bih.biBitCount = 32; // 32 bits (4 bytes) per pixel
    bih.biHeight = -height; // height
    bih.biWidth = width; // width
    bih.biPlanes = 1; // must be 1
    // other fields contains 0

    ofstream f;
    f.open(fileName, ios::binary); // open the file for recording

    f.write((char*)&bfh, sizeof(bfh)); // write the titles
    f.write((char*)&bih, sizeof(bih));

    // Записываем изображение
    for (int i = 0; i < height; i++)
        f.write((char*)colors[i], sizeof(colors[0][0]) * width);

    f.close();
}
#pragma pack (pop)

int main(int argc, char* argv[])
{
    int w = 0, h = 0;
    cout << "Input height in px" << endl;
    cin >> h;
    cout << "Input width in px" << endl;
    cin >> w;

    unsigned int** color = new unsigned int*[h];
    for (int i = 0; i < h; i++)
        color[i] = new unsigned int[w];

```

```

for (int i = 0; i < h; i++)
    for (int j = 0; j < w; j++)
        color[i][j] = 0xFF0000; //to set red color

CreateBmp("test.bmp",color, h, w);

for (int i = 0; i < h; i++)
    delete []color[i];
delete []color;
cout << "test.bmp has been successfully created; Press Enter
to exit";
cin.ignore(2); //
return 0;
}

```

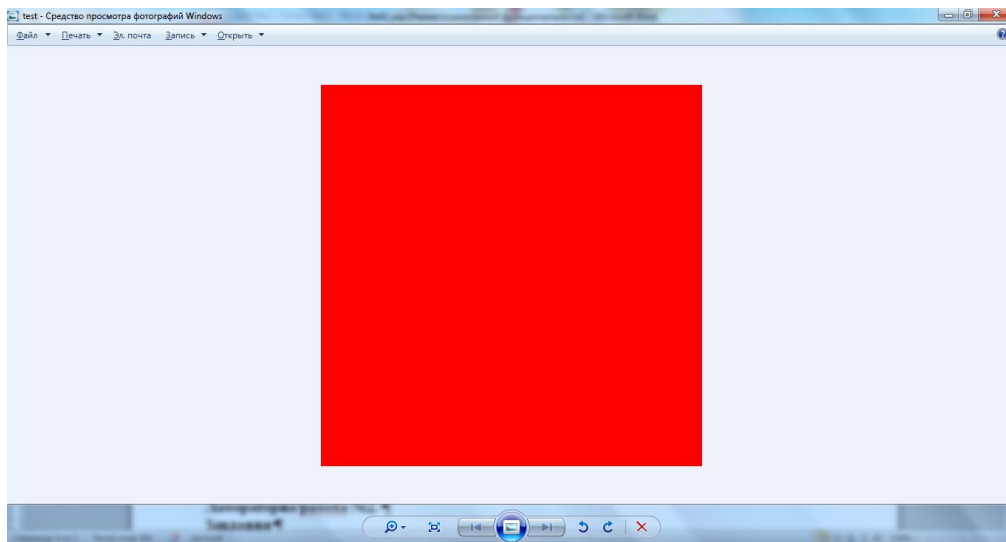


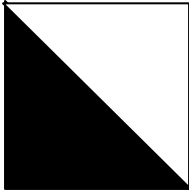
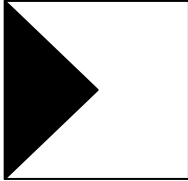
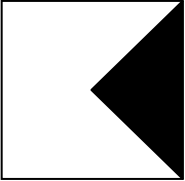
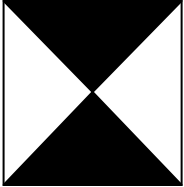
Fig. 3. Saved image in a .bmp file

The Laboratory work №2.

Task

Write a console application in which:

1. The user can set the size of the image.
2. The image is formed as shown below for each variant.
The color of the light and dark parts must be set by the user.
3. The generated image must be saved as a .bmp file.

Variant №1	Variant №2	Variant №3	Variant №4
			
Variant №5	Variant №6	Variant №7	Variant №8
