CSCI 3280 Introduction to Multimedia Systems

Spring 2022, Assignment 1 - ASCII Art (Report)

Student Name: Lai Man Hin SID:1155136167

Introduction

This program has 3 functions:

1. File compression by resizing
2. 8-Level ASCII Art bitmap
3. Colored ASCII Art bitmap (Modified into 16-levels!)

After compile the program and run the program by

```
C:\> cl.exe bonus.cpp bmp.cpp
C:\> bonus yourimage.bmp
```

Enjoy! ;)

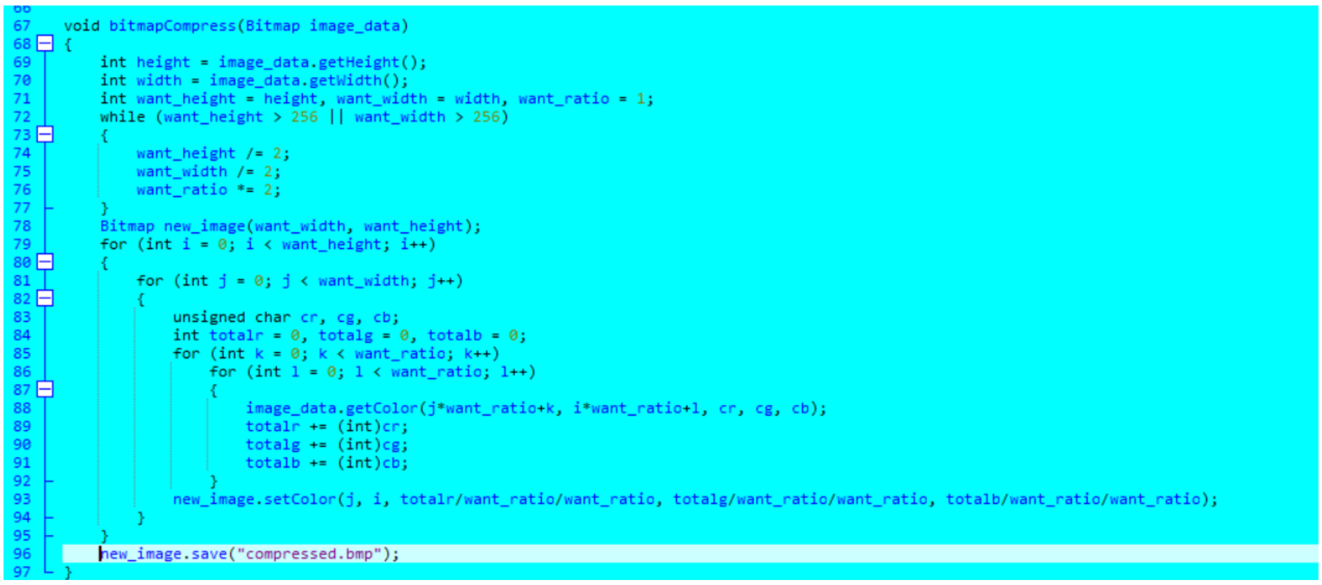
File compression by resizing

If the input .bmp file's height or width is larger than 256 bits, the system will then ask whether want to resize it to be smaller than 256x256 bit image to continue. If no, the program will exit immediately; if yes, the program will keep on smaller the size of the image, by taking half of height and width, until it is smaller than 256x256 bit. The compressed bitmap will be saved as `compressed.bmp` then the program will exit. So please run the command

```
C:\> bonus compressed.bmp
```

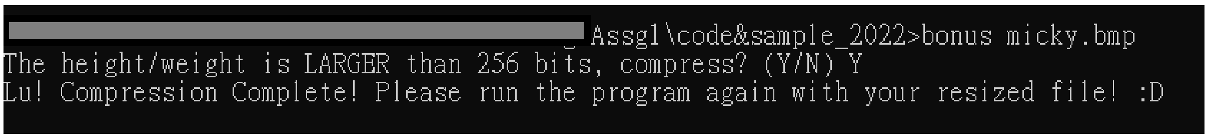
, to enjoy other function!

Screenshot of source code:



```
void bitmapCompress(Bitmap image_data)
{
    int height = image_data.getHeight();
    int width = image_data.getWidth();
    int want_height = height, want_width = width, want_ratio = 1;
    while (want_height > 256 || want_width > 256)
    {
        want_height /= 2;
        want_width /= 2;
        want_ratio *= 2;
    }
    Bitmap new_image(want_width, want_height);
    for (int i = 0; i < want_height; i++)
    {
        for (int j = 0; j < want_width; j++)
        {
            unsigned char cr, cg, cb;
            int totalr = 0, totalg = 0, totalb = 0;
            for (int k = 0; k < want_ratio; k++)
                for (int l = 0; l < want_ratio; l++)
                {
                    image_data.getColor(j*want_ratio+k, i*want_ratio+l, cr, cg, cb);
                    totalr += (int)cr;
                    totalg += (int)cg;
                    totalb += (int)cb;
                }
            new_image.setColor(j, i, totalr/want_ratio/want_ratio, totalg/want_ratio/want_ratio, totalb/want_ratio/want_ratio);
        }
    }
    new_image.save("compressed.bmp");
}
```

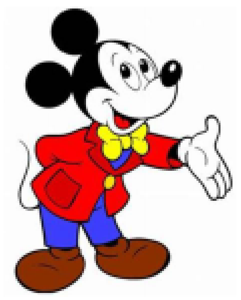
Sample run on command line interface:



```
[redacted]Assg1\code&sample_2022>bonus micky.bmp
The height/weight is LARGER than 256 bits, compress? (Y/N) Y
Lu! Compression Complete! Please run the program again with your resized file! :D
```

Sample input: **`micky.bmp`** (467x583)

Sample output: **`compressed.bmp`** (116x145, 1/4 smaller on both height and width!)



The technique used here is combining multiple pixel's color into one pixel. For example:

Coor(0, 0) has RGB(26, 26, 67)

Coor(0, 1) has RGB(26, 28, 66)

Coor(1, 0) has RGB(26, 28, 67)

Coor(1, 1) has RGB(26, 26, 68)

These 4 pixels will be combined into 1 pixel by taking the mean value of four R, G and B.

In this example the combined pixel has RGB(26, 27, 57).

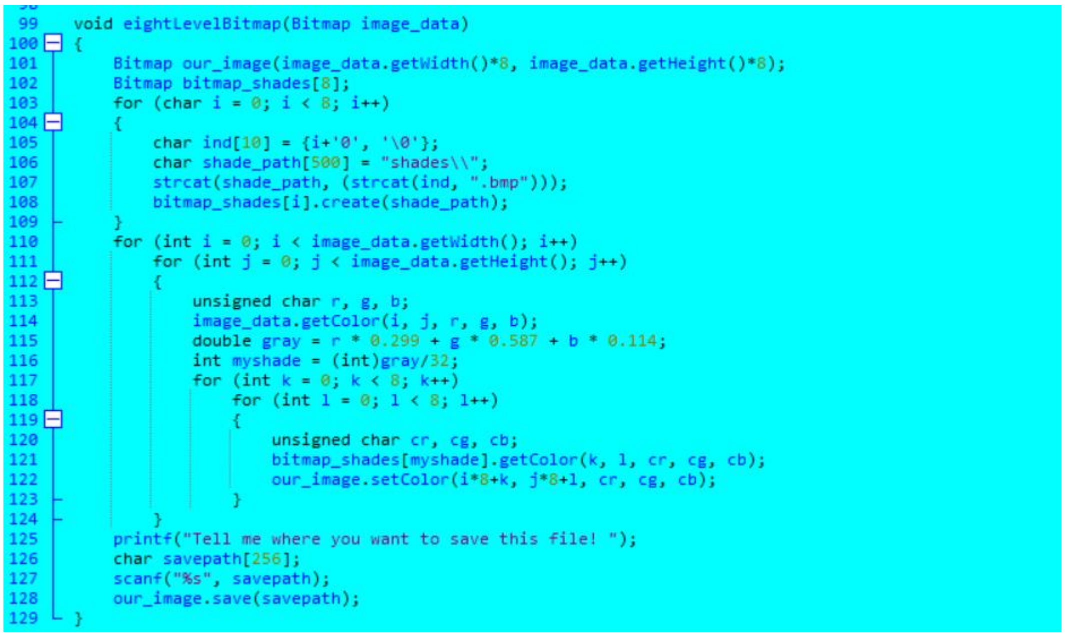
Don’t forget to run the program with the new generated file again to try other functions!

8-Level ASCII Art bitmap

The notepad is not LARGE enough to see almost 200 lines of characters! D: Never mind! The bonus part also provides function to make the ASCII art to be a bitmap file (Note that all the ASCII art bitmap file are stored inside the shades folder). The function will work as follows:

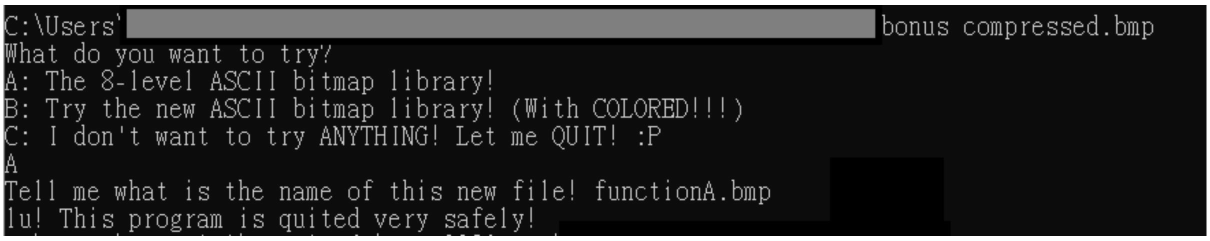
1. Analyze the input file, divide each pixel into 8-level brightness.
2. Print ASCII art bitmap into the new bitmap file pixel by pixel. As each ASCII art bitmap character takes 8x8 pixels in the new bitmap, the final output file will be 8 times LARGER in both width and height (example, an 255x255 input bitmap file will become a 2040x2040 output ASCII Art bitmap file)

Screenshot of source code:



```
void eightLevelBitmap(Bitmap image_data)
{
    Bitmap our_image(image_data.getWidth()*8, image_data.getHeight()*8);
    Bitmap bitmap_shades[8];
    for (char i = 0; i < 8; i++)
    {
        char ind[10] = {i+'0', '\0'};
        char shade_path[500] = "shades\\";
        strcat(shade_path, (strcat(ind, ".bmp")));
        bitmap_shades[i].create(shade_path);
    }
    for (int i = 0; i < image_data.getWidth(); i++)
        for (int j = 0; j < image_data.getHeight(); j++)
        {
            unsigned char r, g, b;
            image_data.getColor(i, j, r, g, b);
            double gray = r * 0.299 + g * 0.587 + b * 0.114;
            int myshade = (int)gray/32;
            for (int k = 0; k < 8; k++)
                for (int l = 0; l < 8; l++)
                {
                    unsigned char cr, cg, cb;
                    bitmap_shades[myshade].getColor(k, l, cr, cg, cb);
                    our_image.setColor(i*8+k, j*8+l, cr, cg, cb);
                }
        }
    printf("Tell me where you want to save this file! ");
    char savepath[256];
    scanf("%s", savepath);
    our_image.save(savepath);
}
```

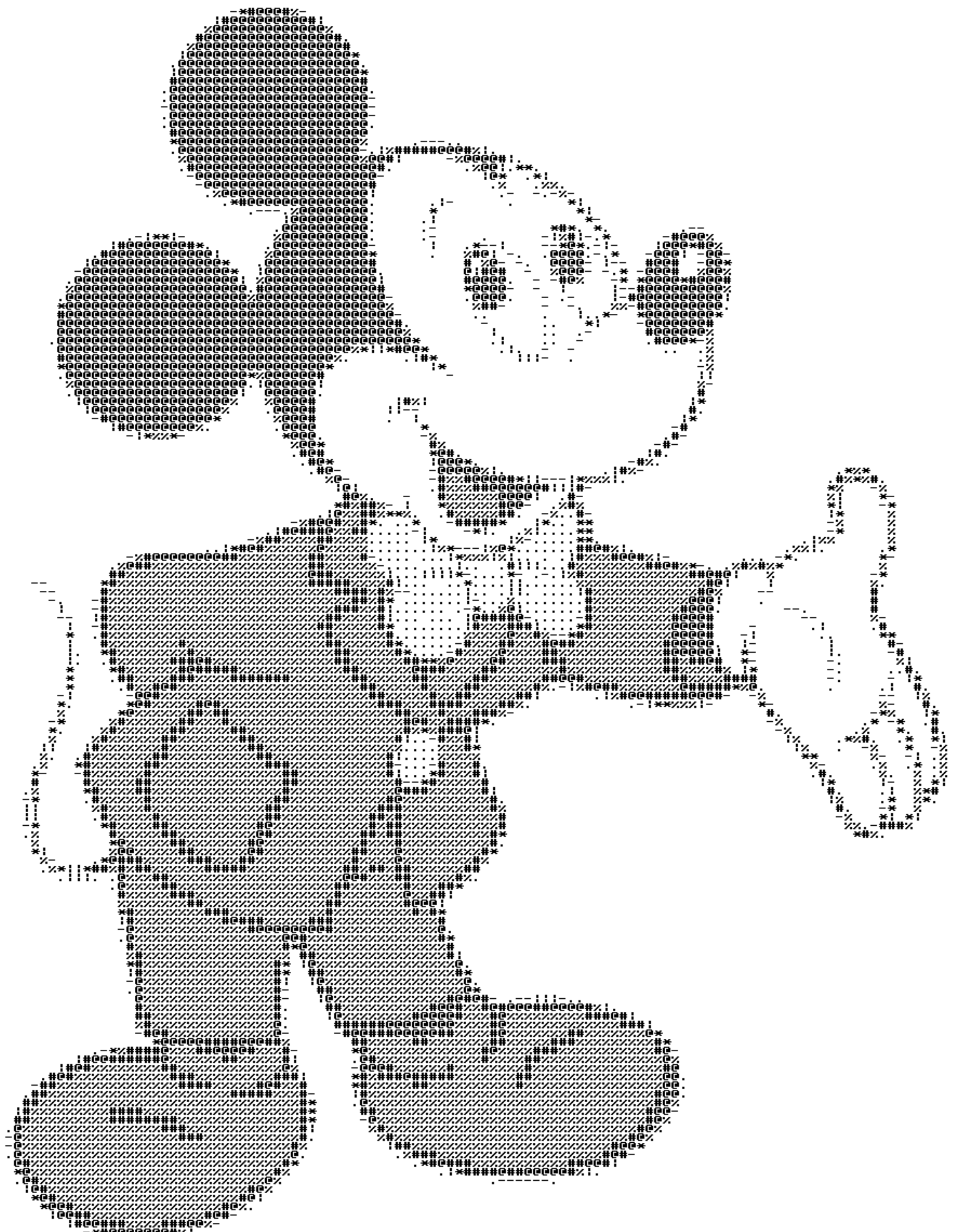
Sample run:



```
C:\Users\[redacted]bonus compressed.bmp
What do you want to try?
A: The 8-level ASCII bitmap library!
B: Try the new ASCII bitmap library! (With COLORED!!!)
C: I don't want to try ANYTHING! Let me QUIT! :P
A
Tell me what is the name of this new file! functionA.bmp
lu! This program is quited very safely!
```

After run the program with proper bitmap file, input ‘A’ to enter the function. And finally input the output file name.

Output file: **`functionA.bmp`**



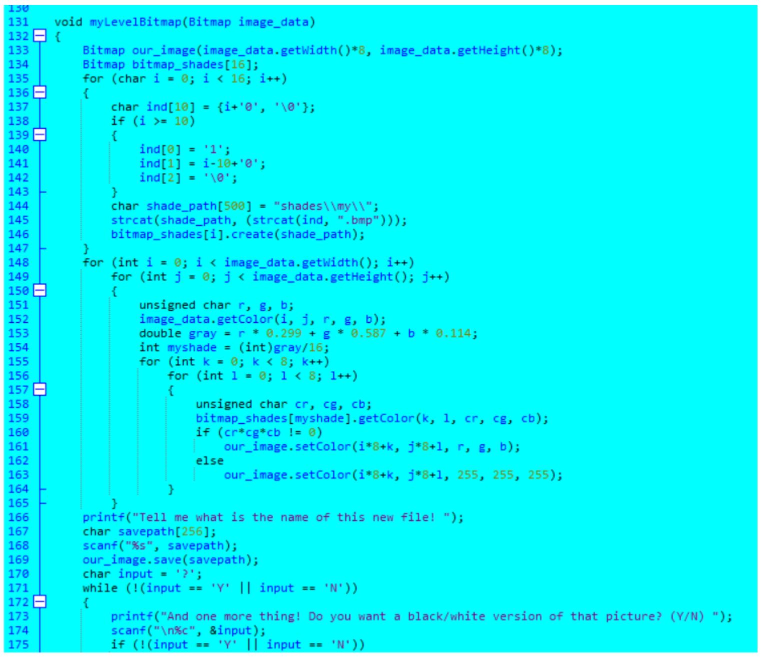
Technique used: It is similar to the **`ascii.cpp`** ones. However, instead of saving the assigned characters into a 2D-character array and output it into a file, this function copies the source ASCII art bitmap file to the corresponding pixels of the output file. For example, (65, 12) is detected as the brightness level 4 (128-160 in gray scale), the function will copy all pixels in **`shades\4.bmp`**, into ([520,527], [96, 103]).

Colored ASCII Art bitmap (Modified into 16-levels!)

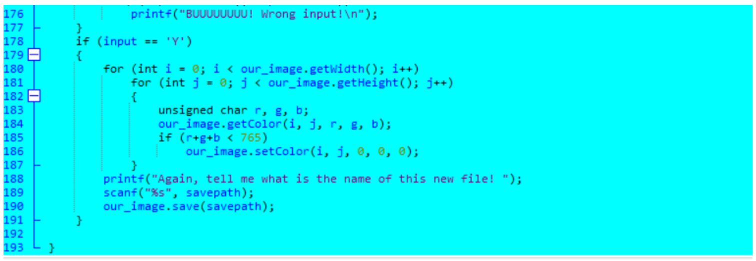
Instead of only black-and-white-only ASCII Art bitmap pictures, there also provide Colored ASCII Art bitmap. At the same time, the ASCII Art bitmap are also modified into 16 levels so as to provide better quality of output images. (Note that all the ASCII art bitmap file are stored in a folder which is inside the shades folder. i.e. **`shades\my\`**) The function will work as follows:

1. Analyze the input file, divide each pixel into 16-level brightness (similar to the previous function)
2. Print ASCII art bitmap into the new bitmap file pixel by pixel. When there is a written pixel (aka. white pixel) is detected in the ASCII art bitmap, the output written pixel will be set to be the original pixel color instead of white.
3. After save the new Colored ASCII Art bitmap file successfully. The program will also ask whether the user also wants the black-and-white version of that file (In 16-levels, own bitmap)

Screenshot of source code:



```
void myLevelBitmap(Bitmap image_data)
{
    Bitmap our_image(image_data.getWidth()*8, image_data.getHeight()*8);
    Bitmap bitmap_shades[16];
    for (char i = 0; i < 16; i++)
    {
        char ind[10] = {i+'0', '\0'};
        if (i >= 10)
        {
            ind[0] = '1';
            ind[1] = i-10+'0';
            ind[2] = '\0';
        }
        char shade_path[500] = "shades\\my\\";
        strcat(shade_path, (strcat(ind, ".bmp")));
        bitmap_shades[i].create(shade_path);
    }
    for (int i = 0; i < image_data.getWidth(); i++)
        for (int j = 0; j < image_data.getHeight(); j++)
        {
            unsigned char r, g, b;
            image_data.getColor(i, j, r, g, b);
            double gray = r * 0.299 + g * 0.587 + b * 0.114;
            int myshade = (int)gray/16;
            for (int k = 0; k < 8; k++)
                for (int l = 0; l < 8; l++)
                {
                    unsigned char cr, cg, cb;
                    bitmap_shades[myshade].getColor(k, l, cr, cg, cb);
                    if (cr*cg*cb != 0)
                        our_image.setColor(i*8+k, j*8+l, r, g, b);
                    else
                        our_image.setColor(i*8+k, j*8+l, 255, 255, 255);
                }
        }
    printf("Tell me what is the name of this new file! ");
    char savepath[256];
    scanf("%s", savepath);
    our_image.save(savepath);
    char input = '?';
    while (!(input == 'Y' || input == 'N'))
    {
        printf("And one more thing! Do you want a black/white version of that picture? (Y/N) ");
        scanf("\n%c", &input);
        if (!(input == 'Y' || input == 'N'))
```

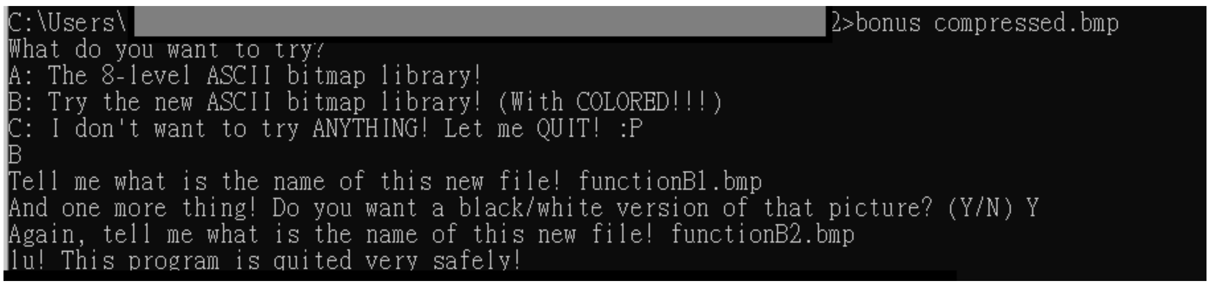


```
            printf("BUUUUUUUU! Wrong input!\n");
    }
    if (input == 'Y')
    {
        for (int i = 0; i < our_image.getWidth(); i++)
            for (int j = 0; j < our_image.getHeight(); j++)
            {
                unsigned char r, g, b;
                our_image.getColor(i, j, r, g, b);
                if (r+g+b < 765)
                    our_image.setColor(i, j, 0, 0, 0);
            }
        printf("Again, tell me what is the name of this new file! ");
        scanf("%s", savepath);
        our_image.save(savepath);
    }

}
```

Sample run:

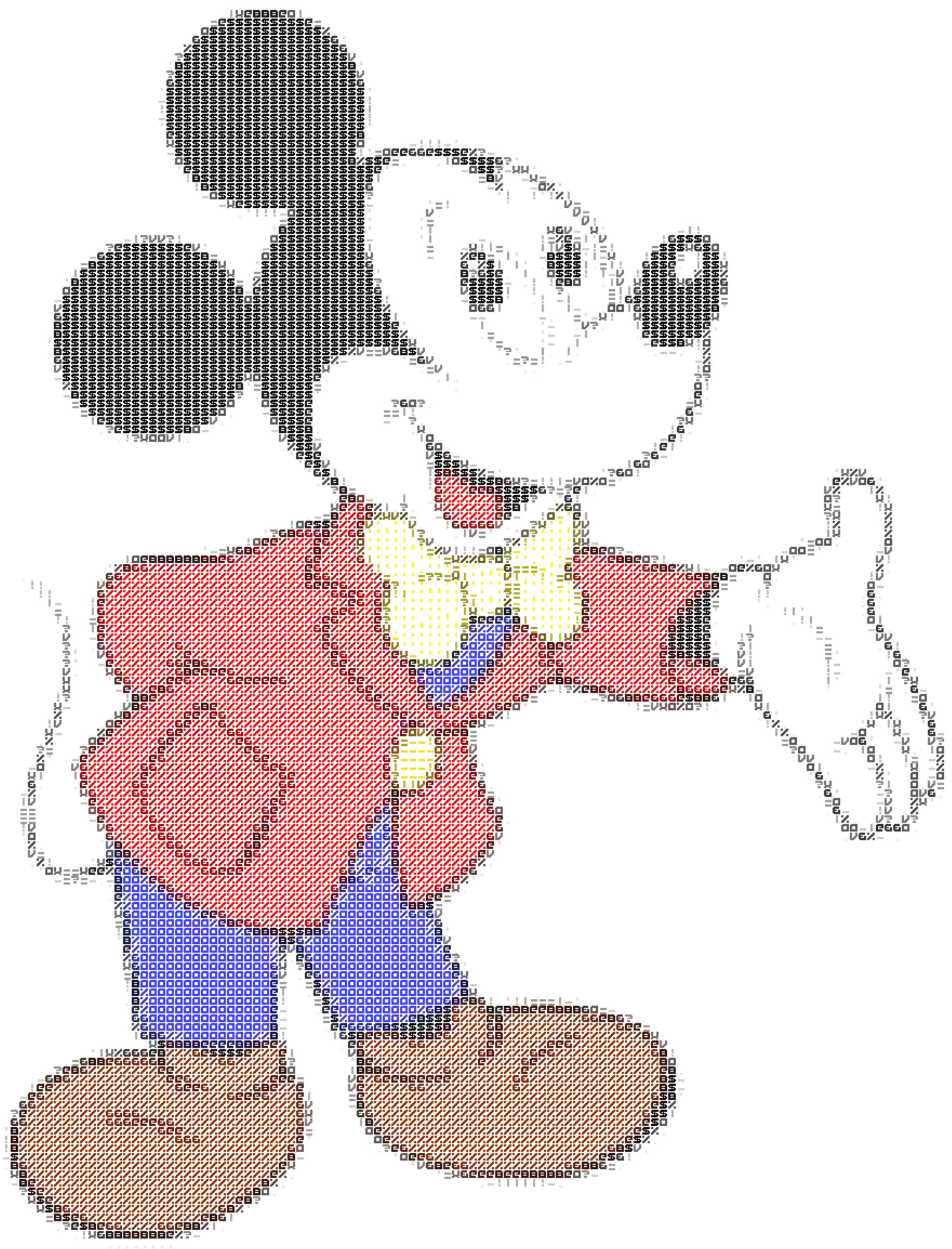
After run the program with proper bitmap file, input 'B' to enter the function. And finally input the output file name. If you want the black-and-white version, type 'Y' and input the name of output file for saving, type ‘N’ to exit directly.



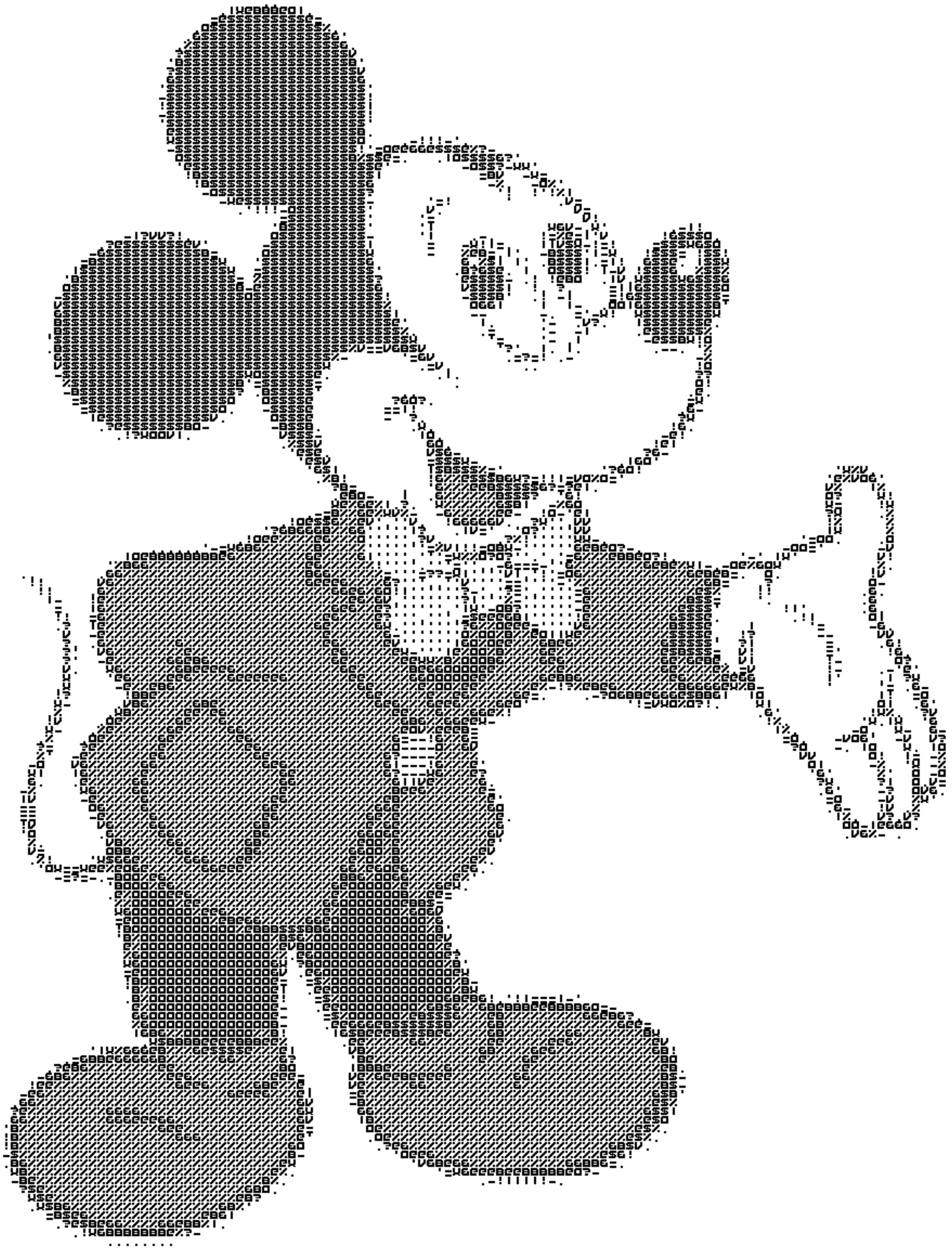
```
C:\Users\[redacted]2>bonus compressed.bmp
What do you want to try?
A: The 8-level ASCII bitmap library!
B: Try the new ASCII bitmap library! (With COLORED!!!)
C: I don't want to try ANYTHING! Let me QUIT! :P
B
Tell me what is the name of this new file! functionB1.bmp
And one more thing! Do you want a black/white version of that picture? (Y/N) Y
Again, tell me what is the name of this new file! functionB2.bmp
lu! This program is quited very safely!
```

(Continue next page)

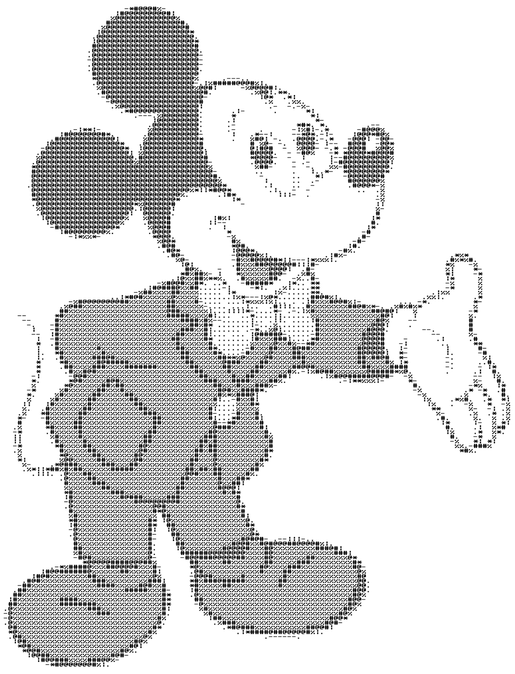
Output1: function2A.bmp

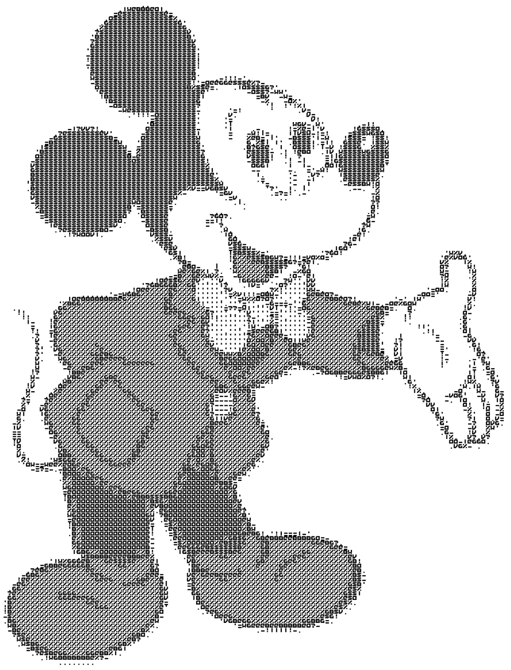


Output2: function2B.bmp



Comparison between the 8-bit level mapping (**`functionA.bmp,`** on left) and 16-bit level mapping (**`function2B.bmp,`** on right)





The red color in clothes and the blue color in the trousers are distinguished well in 16-bit level mapping one!

Technique used: Similar to the previous function, but a 16-level mapping level is used instead of 8-level. Also, for **`functionB1.bmp`** one, instead of assigning the black or white color, it uses the original bitmap's pixel color. (See line 160-163)

References

For the font used in another bitmap:
https://dragon.style/system/media_attachments/files/000/202/613/original/71c1fcbb7b70661b.png

This website gives me idea how to modify the mapping levels:
http://paulbourke.net/dataformats/asciiart/