CSCI3280 Assignment 1 enhancement part report

ascii1.cpp:1. Compile the program with “cl ascii1.cpp bmp.cpp” in the command prompt.

2. Enter in the format of “ascii1 <bmp file> <filter size> <output bmp file name>

eg. ascii1 micky.bmp 4x4 micky2.bmp

function: 1. pixelate the image with given filter size(eg 4x4)

source code related:

```
    Bitmap newimage(width, height);

    unsigned char  tr,tg,tb;
    int r,g,b;
    int w_r, h_r;
    int w_scale = width / filter_w;
    int h_scale = height / filter_h;
    // printf("filter_h: %d filter_w: %d\n",filter_h,filter_w);
    // printf("w_scale :%d h_scale:%d",w_scale,h_scale);
    w_r = width % filter_w;
    h_r = height % filter_h;

    for (int i = 0; i < h_scale; i++) {
        for (int j = 0; j < w_scale; j++) {
            r = 0;
            g = 0;
            b = 0;
            for (int k = i*filter_h; k < (i+1)*filter_h; k++) {
                for (int l = j*filter_w; l < (j+1)*filter_w; l++) {
                    image_data.getColor(l,k,tr,tg,tb);
                    r += tr;
                    g += tg;
                    b += tb;
                }
            }
            r = (int) round( (double) r / (filter_h*filter_w));
            g = (int) round( (double) g / (filter_h*filter_w));
            b = (int) round( (double) b / (filter_h*filter_w));
            for (int k = i*filter_h; k < (i+1)*filter_h; k++)
                for (int l = j*filter_w; l < (j+1)*filter_w; l++) {
                    newimage.setColor(l,k,r,g,b);
```

```
                }

        }
    }
    for (int i = h_scale*filter_h; i < h_scale*filter_h + h_r; i++)

        for (int j = 0; j < width; j++) {

            image_data.getColor(j,i-1,tr,tg,tb);

            newimage.setColor(j,i,tr,tg,tb);
        }

    for (int j = w_scale*filter_w; j < w_scale*filter_w + w_r; j++)

        for (int i = 0; i < height; i++) {

            image_data.getColor(j-1,i,tr,tg,tb);
            newimage.setColor(j,i,tr,tg,tb);

        }
    newimage.save(argv[3]);
```

Algorithm: Average the color in the given filter size and set the corresponding pixels of the new image. Then set the color of the new image to the original where the filter does not cover(eg. 100x100 bmp image, filter with 3x3, there should be a whole column and whole row that the filter could not implement on it). Finally, save the bmp file of given name.

Sample run: command line running: ascii1 micky.bmp 4x4 micky1.bmp
Output: a bmp file named micky1.bmp is pixelated from micky.bmp

(micky1.bmp : size unchanged)

ascii3.cpp:1. Compile the program with “cl ascii2.cpp bmp.cpp” in the command prompt.

2. Enter in the format of “ascii2 <bmp file name> <resize dimension> <save bmp file name> <html file name>

eg ascii3 mario.bmp 79x68 mario3.bmp mario3.html

function: 1. resize to specific height and width (can only accept the resize dimension that is less than or equal to the original dimension eg original 100x100 bmp image. It could not resize to height/width greater than 100) and save it as bmp.

source code related:

```
    w_scale = width / (double) new_width;
    h_scale = height / (double) new_height;
    printf("New_height is : %d \nNew_width is : %d \n",new_height,new_w
idth);

    Bitmap newimage(new_width, new_height);

    double merge;
    int r,g,b;
    unsigned char tr,tg,tb;
    printf("w_scale : %.3f h_scale : %.3f\n",w_scale,h_scale);
    for (int i = 0; i < new_height; i++) {
        for (int j = 0; j < new_width; j++) {
            r = 0;
            g = 0;
            b = 0;
            for (int k = (int) round(i*h_scale); k < (int) round((i+1)*
h_scale); k++)
                for (int l = (int) round(j*w_scale); l < (int) round((j
+1)*w_scale); l++) {
                    image_data.getColor(l,k,tr,tg,tb);
                    r += tr;
                    g += tg;
                    b += tb;
                }
            r = (int) round( r / ((round((i+1)*h_scale)-
round(i*h_scale)) * (round((j+1)*w_scale)-round(j*w_scale)))) ;
            g = (int) round( g / ((round((i+1)*h_scale)-
round(i*h_scale)) * (round((j+1)*w_scale)-round(j*w_scale))));
            b = (int) round( b / ((round((i+1)*h_scale)-
round(i*h_scale)) * (round((j+1)*w_scale)-round(j*w_scale))));
            newimage.setColor(j,i,r,g,b);
        }
    }
newimage.save(argv[3]);
```

2. output the image in colored ascii code in html format

source code related:

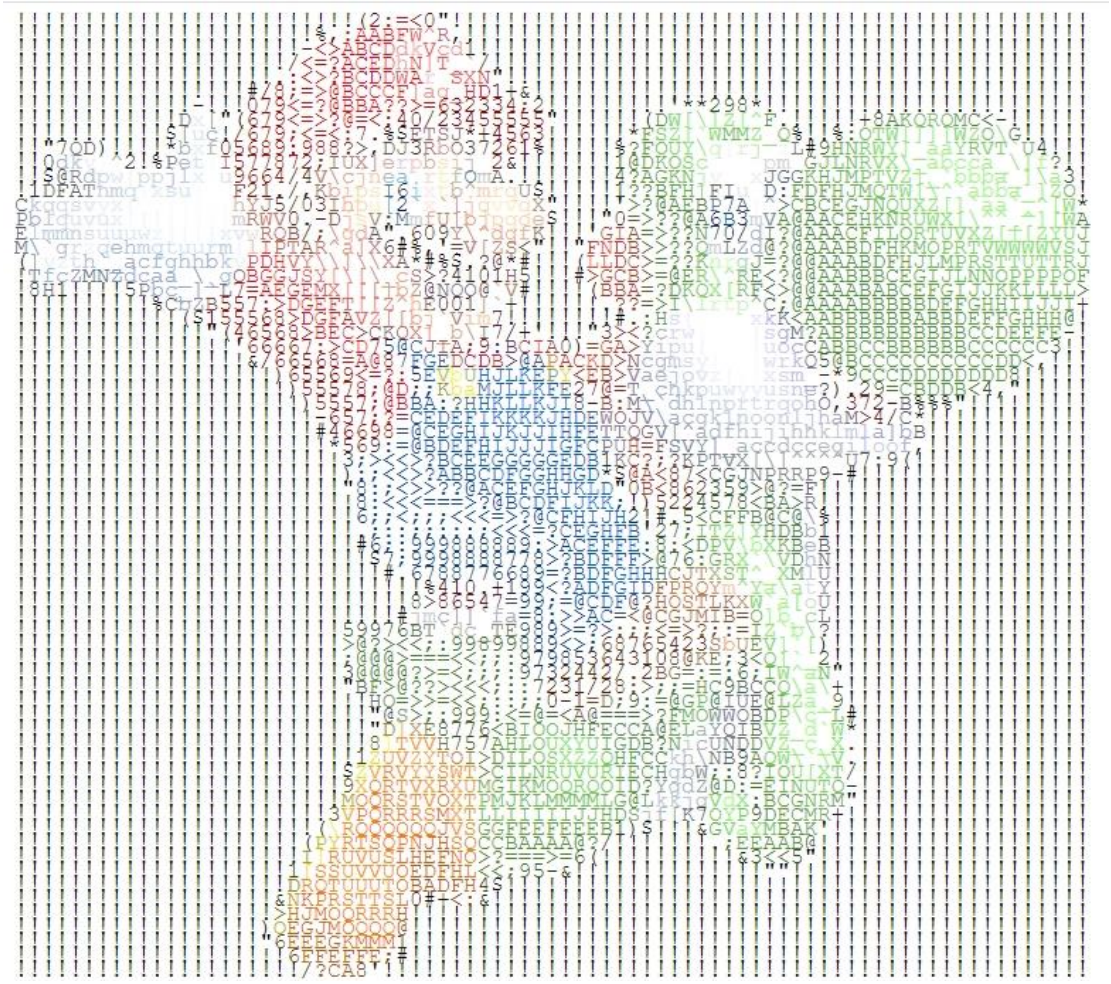
```
FILE *html;
    int temp;
    char color[7];
    char c;
    unsigned char rgb[4];
    html = fopen( argv[4] , "w");
    fputs("<html>\n<body bgcolor = #ffffff>\n<font face = \"Courier New
\">\n",html);
    fputs("<p style = \" line-height:0.625\">", html);
    for (int i = 0; i < new_height; i++) {
        for (int j = 0; j < new_width; j++) {
            newimage.getColor(j,i,rgb[0],rgb[1],rgb[2]);
            fputs("<font color = #",html);
            int x = 0;
            strcpy(color,"");
// Convert the color to hexadecimal number
            while (3>x) {
            temp = rgb[x] % 16;
            if (temp >= 10) {
                color[x*2 + 1] = temp + 55;
            }
            else color[x*2 + 1] = temp + 48;
            temp = rgb[x] / 16 % 16;
            if (temp >= 10) {
                color[x*2] = temp + 55;
            }
            else color[x*2] = temp + 48;

            x++;
            }
            fprintf(html, "%.6s",color);

            fputc('>',html);
// Map the character in the ascii table according to the color value.
            char temp_c = round(33+(rgb[0]+rgb[1]+rgb[2])/((double)765/
93));
            fputc(temp_c,html);
            fputs("</font>",html);
```

```
        }
        fputs("<br>\n",html);
    }
    fputs("</body>\n</html>",html);
    fclose(html);
```

Algorithm: Get the filter scale by dividing the original height/width to new height/width (stored as double). Let the position of x and y be j and i. Average the color in the original image by looping through the pixels between (round(i*height_scale) and round((i+1)*height_scale)) and (round(j*width_scale) and round((j+1)*width_scale)). Then set the color of the image of position (j,i). After resizing, it could save the resized image into bmp file with name <save bmp file name>. When the <html file name> is entered, it will process to enter the character and color one pixel by one with the font Courier New in the html file. The line-height is set to 0.625 in order to make the html text to close to 1:1 scale. The character would base on the color value of the pixel ie (total 255x3 = 765) and map the color value to the ascii code from 33 to 126. The color value of RGB would be converted to hexadecimal first in order to write to the html file by short division method.

Sample run: command line running: ascii3 mario.bmp 79x68 mario3.bmp mario3.html
Output: 1. A resized bmp file entitled mario3.bmp with dimension (79x68)

2. A html file entitled mario3.html (same dimension as 1) (screencap)



ascii4.cpp:1. Compile the program with “cl ascii2.cpp bmp.cpp” in the command prompt.

2. Enter in the format of “ascii4 <s/p> <bmp file name> <resize dimension> <save bmp file name>

e.g ascii4 s micky.bmp 77x69 micky4.bmp

3. Note that even when any character entered other than s in the column of <s/p> It will still use the print mapping function that same as the basic part.

function: 1. Resize the image into specific width and height that are neither greater than its original.

(Sample code same as **ascii3.cpp**’s)

2. Map the luminance of the color pixel to 8 ascii characters and according to the bmps of these characters, save the resized image as bmp image.

Sample code:

-Reading the bmps of the ascii code provided in the shades folder

```
Bitmap ascii[8] = {("./shades/0.bmp"),("./shades/1.bmp"),("./shades/2.b
mp"),("./shades/3.bmp"),("./shades/4.bmp"),("./shades/5.bmp"),("./shade
s/6.bmp"),("./shades/7.bmp")};
```

-Obtain luminance:

```
*(quant + i*width + j) = floor((0.299*r + 0.587*g + 0.114*b)/(256/8));
```

-Set the color of the bmp file and save it

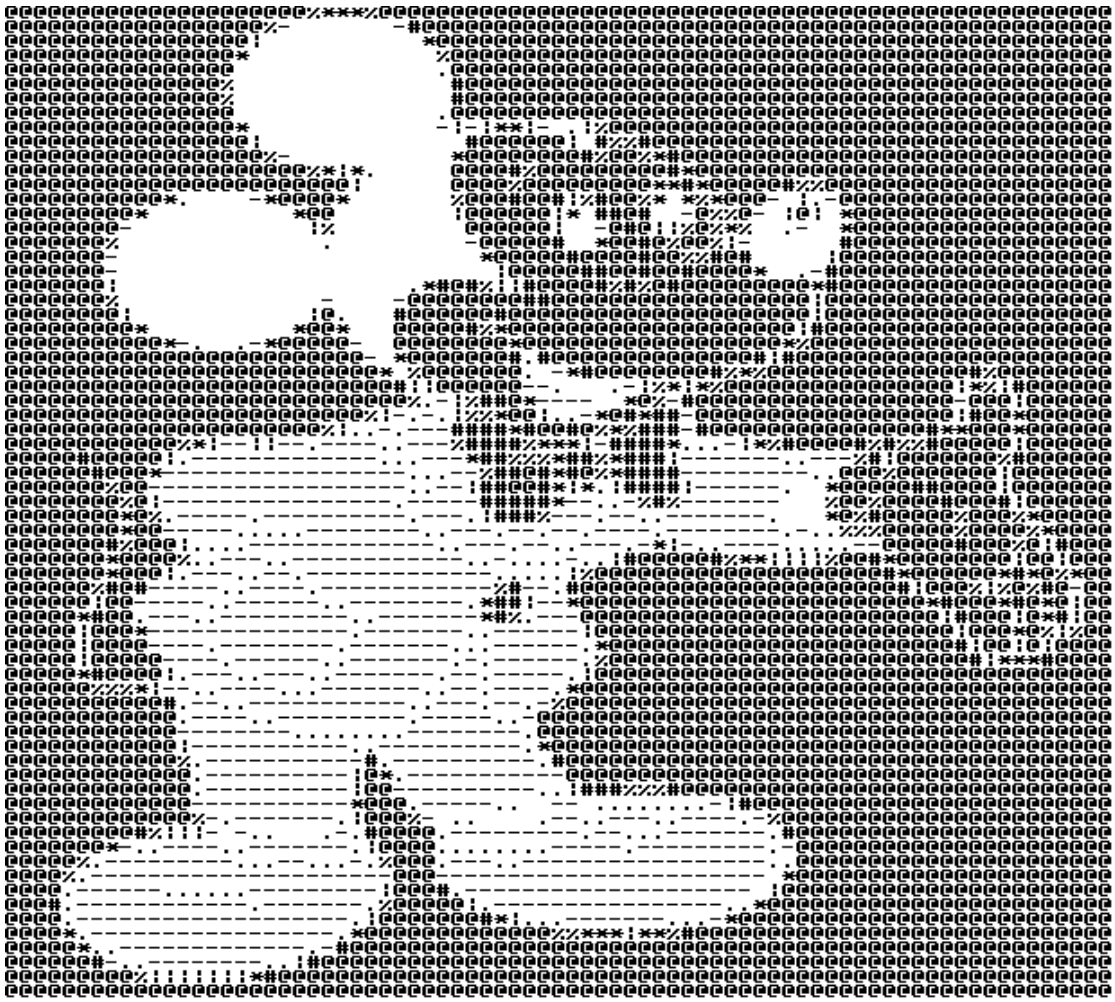
```
    Bitmap ascii_image(8*new_width,8*new_height);
    for (int i = 0; i < new_height; i++) {
        for (int j = 0; j < new_width; j++) {
            int temp = *(quant + i*width + j);
            if (argv[1][0] == 's') temp = 7 - temp;
            for (int k = i*8; k < (i+1)*8; k++) {
                for (int l = j*8; l < (j+1)*8; l++) {
                    ascii[temp].getColor(l - j*8 ,k - i*8 ,tr,tg,tb);

                    ascii_image.setColor(l,k,tr,tg,tb);
                }
            }
        }
    }
    ascii_image.save(argv[4]);
```

Algorithm: The resize algorithm is same as ascii3.cpp. Then it creates bitmap image with 8 times width and height that of the bmp file entered. The program loads the 8 bmp files from the shader folder into an array of bitmaps. Then fill in the enlarged image with these ascii bmp files according to the quantization in the basic part. The one pixel of the resized original image is equal to the 8x8 pixels of the required image, where 8x8 pixels colors come from the bmps of the ascii code according to the quantized value. Finally, save the new image as bmp file.

Sample run: command line running: ascii s micky.bmp 77x69 micky4.bmp

Output: A resized bmp file entitled micky4.bmp that use the bmps of the ascii characters with dimension 77x69

Note: As the argv[1][0] is ‘s’. The mapping with the ascii bmp file would be 7 minus the quantized value instead of just the quantized value like the basic part.



Sample run 2: command line running: ascii4 p micky.bmp 77x69 micky5.bmp

Output: A resized bmp file entitled micky4.bmp that use the bmps of the ascii characters with dimension 77x69.