Middle East Technical University | Department of Computer Engineering

CENG 140

C Programming

Spring 2021
Take Home Exam 1

Ataberk Donmez
ataberk@ceng.metu.edu.tr
Due date: May 11, 2021, Tuesday, 23:55

Introduction

This homework consists of two parts:

1. **In the first part**, you will fill part of an image with a color.
2. **In the second part**, you will copy an area of an image and paste it on another area on the same image.
3. **In the third part**, you will copy, rotate and then paste it on another area on the same image.

Part 1 (40 points)

In this part, you will implement an operation similar to the "Bucket Fill" of the well-known graphics editor Paint. This operation includes three main steps. First, you will select a color. Next, you will be given a pixel coordinate on an image. Then finally, you will change the color of the contiguous area that is the same color as the pixel you were given the coordinates of, to the selected color.

Specifications

1. The image dimensions are fixed (25x25). So that you can store it in a 2-dimensional array with a fixed length (i.e. 25). Some examples on this homework text go through a 5x5 image to make it simpler to understand the basics of the homework. However, all the images that will be used for grading all 3 parts of the assignment will be 25x25.
2. The image consists of $25 * 25 = 625$ pixels, and each pixel is represented by an integer value in the range $[0, 7]$, corresponding to a color. The colors and their corresponding numbers are as follows (can also be seen in Figure 1):

- Red: 0
- Yellow: 1
- Green: 2
- Magenta: 3
- White: 4
- Blue: 5
- Black: 6
- Cyan: 7

3. There is an imaginary palette around the image. This palette can be used to select a color and it is created by extending the edges of the image in all directions and dividing the space into 8 non-overlapping regions. Each region corresponds to a specific color. The edges extend to infinity, so these 8 regions are infinite as well. Check Figure 1 for more clarification and to see which color each region corresponds to.

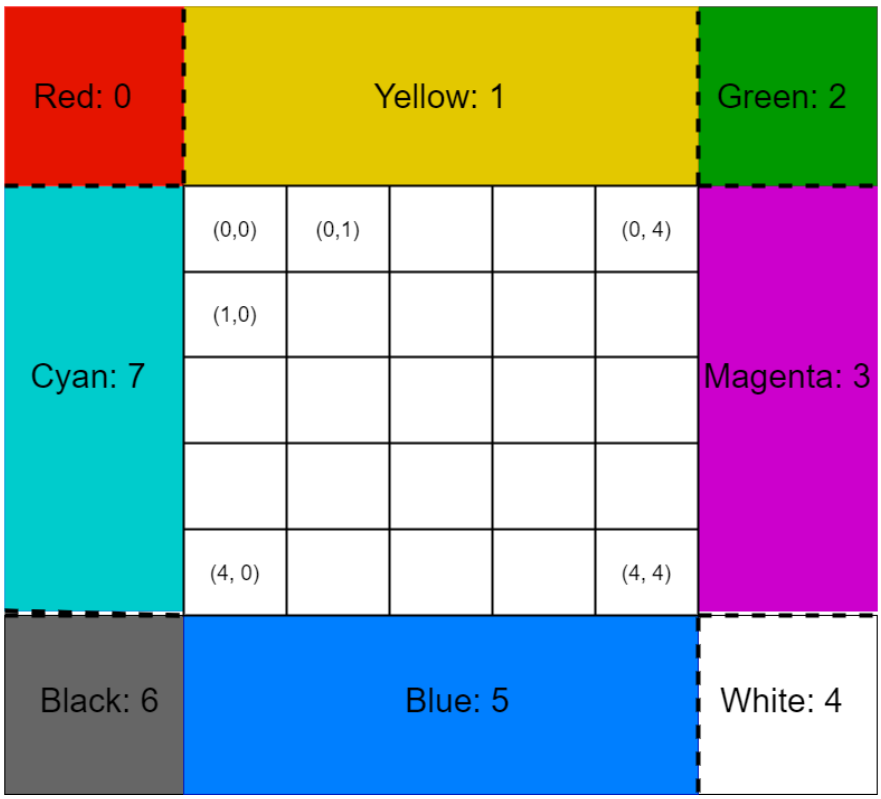


Figure 1: A 5x5 image in the middle and the imaginary palette around it, visualized.

4. The first input you will be given is the image itself. It will be given as 25 integer values each line (with spaces in between) for 25 lines. Remember that each integer value corresponds to the color of a pixel. You need to store the color value for each pixel in the correct position in a 2-dimensional array with a fixed length. Example input for the 5x5 image in Figure 2(a) is

given below.

Example 5x5 image input:

```
0 0 1 2 0
7 0 0 1 2
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
```

5. The second input you will be given is the character `F` denoting fill operation (discussed in this part of the assignment). If the character is `P` or `R` that means a different operation (see part 2 and part 3).

6. The third input you will be given is the row and column coordinates for selecting a color. The only thing we know about the range of these two values is that they are integers. In the case that these coordinates fall on a pixel on the image, you will select this pixel's color. If the coordinates fall outside the image, then you will select the color according to the 8 regions around the image. Notice that the image coordinates start from $(0, 0)$. Below are some examples with a 5x5 image.

   Example color select coordinates and the selected color (assuming the image is the one in Figure 2(a)):
   Input `0 2` would select the color yellow (corresponding integer value: 1).
   Input `-1 -1` would select the color red (corresponding integer value: 0).
   Input `3 8` would select the color Magenta (corresponding integer value: 3).

7. The final input you will be given is the row and column coordinates of the initial pixel you will start the fill operation. You will fill the contiguous area that is the same color as the initial pixel. The given coordinates will always be coordinates of a pixel on the image. The color you will use to fill this area is determined by the third input you were given. While finding the contiguous area, you need the check 8 neighboring pixels in all directions (north, east, south, west, northeast, southeast, northwest, southwest). **You need to implement this part recursively.**

8. After the fill operation is complete, you will print the final image. Similar to the input, there should be 25 pixel values in each line for 25 lines. There should be a space character after every color value (including the last value in a line), and there should be a new line character after every line (including the last line).

   Now let's go through an example with a 5x5 image together to understand better.

   Input:

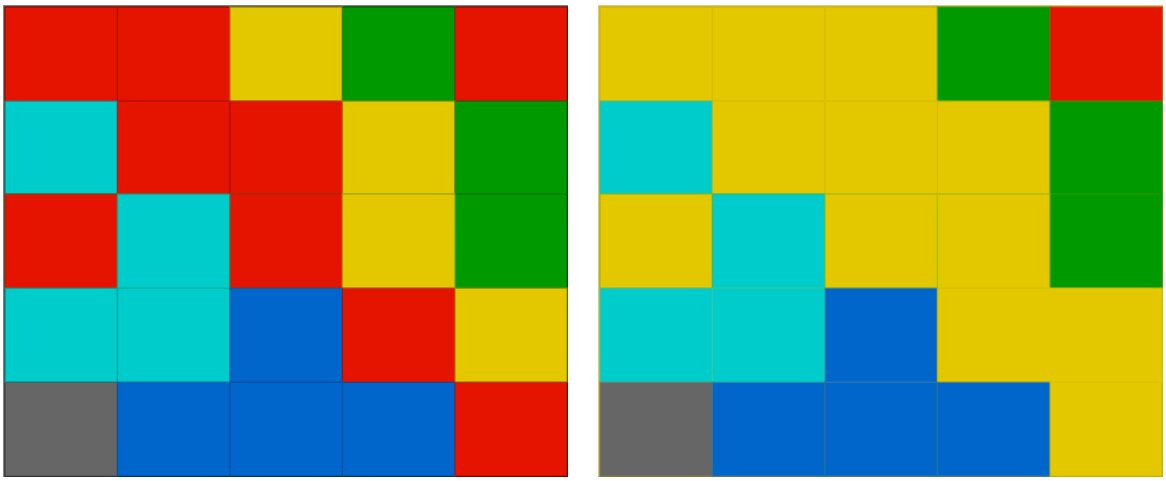
```
0 0 1 2 0
7 0 0 1 2
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
F
0 2
1 2
```

Output:

```
1 1 1 2 0
7 1 1 1 2
1 7 1 1 2
7 7 5 1 1
6 5 5 5 1
```

Explanation:

- The first 5 lines of input are the pixel colors for each pixel on the image. Notice that each line has 5 values separated by a space character.
- Second input is `F` so the operation we are going to do is fill.
- The third input (on the 7'th line) is the coordinates for picking the color. We see that the coordinate (0, 2) falls on the image and we take the color of the pixel on row 0, column 2 (which is yellow).
- Next, we take coordinates of the pixel we will start the filling. In this case, it is the pixel on row 1, column 2. This is a red pixel. So the contiguous red area the pixel is in, should be painted yellow. We continue with a recursive approach.
- Notice that if the color picking coordinates were `(-50, 2)`, the output would be exactly the same. Because according to the regions around the image, we would choose yellow again.
- The input and the output images for this example can be seen in Figure 2.



(a) Before fill operation (b) After fill operation

Figure 2: Before and after the fill operation. The color is selected from $(0, 2)$, and filling is started from pixel $(1, 2)$.

Hints

- You can define macros for maximum width and height of the image and use them anywhere you need, including the declaration of the 2-dimensional fixed length array you will use to store the pixel colors. Doing this would make it easier to debug your code with smaller input images by changing the macro definitions only. Just don't forget to change the macro definitions to 25 before submitting your code.
- You can save your inputs to a file and use input/output redirection instead of typing your inputs every time manually. For example if the name of the input file is `input.txt` and your executable is `the1`, in your terminal you can do: `./the1 < input.txt > output.txt` to write your output to the file `output.txt`.

Part 2 (20 points)

In this part you will be given two points that define a rectangular area of the image, and you are going to copy and paste this area onto another.

Specifications

1. The first input you will be given is the image. Image format is the same as in part 1 (25x25 with values between [0, 7] for each pixel).
2. Second input you will be given is the character `P`. This means we will do a copy-paste operation (instead of fill as in part 1).
3. The next two inputs you will be given are the locations of 4 pixels that define the rectangular area you will copy from and the rectangular area you will paste onto. First two pixels are the diagonally opposite corners of the copy area and the second two pixels are the diagonally opposite corners of the paste area. The format is given below.

   Input format:
   `<copy_point_1_row> <copy_point_1_col> <copy_point_2_row> <copy_point_2_col>`
   `<paste_point_1_row> <paste_point_1_col> <paste_point_2_row> <paste_point_2_col>`
   Input example:
   ```
   2 2 0 0
   3 3 5 5
   ```
4. This is all the information you need to copy and paste an area! After pasting onto the target area, you will print the modified image (same output format as in part 1) and the number of pixels affected (whose color is changed) by the paste operation. After printing this value as an integer, do not forget to put a new line at the end!
5. The pixels, whose coordinates you are given as input, are included in copy-paste operations.
6. Shape, size and orientation of the two areas will be the same, both areas will be inside the image completely and they will not overlap (i.e. no erroneous input!).

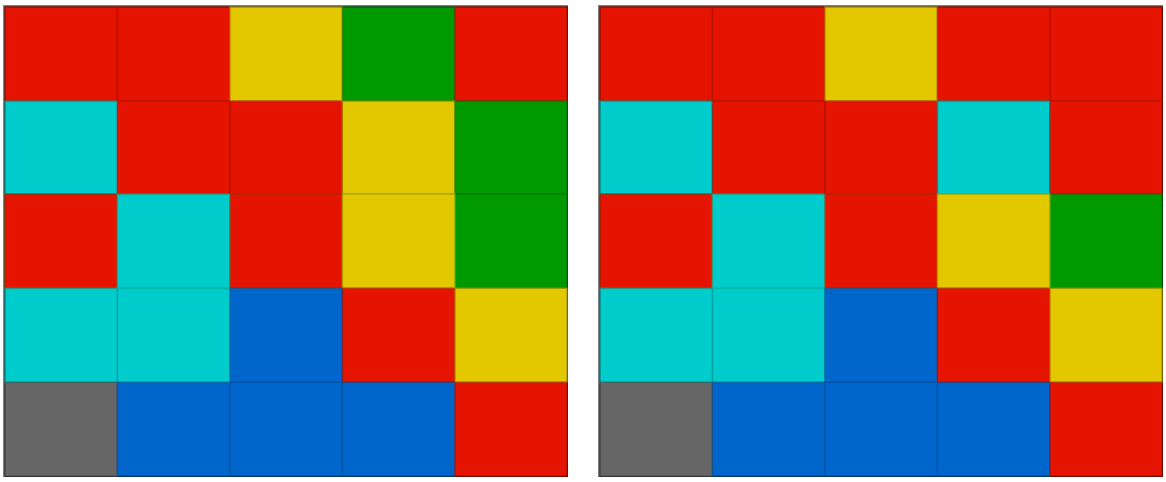
   Let's see an example!

Input:

```
0 0 1 2 0
7 0 0 1 2
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
P
1 1 0 0
0 3 1 4
```

Output:

```
0 0 1 0 0
7 0 0 7 0
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
3
```



(a) Before copy-paste operation (b) After copy-paste operation

Figure 3: Before and after the copy-paste operation.

Explanation:

- Input `1 1 0 0` gives us the two pixels at (0, 0) and (1, 1) and these two pixels together define the area with the following pixels: (0,0), (0, 1), (1, 0), (1, 1). Notice that we know that they are diagonally opposite corners but we do not know anything else!
- Similarly the area we will paste onto are defined by the pixels (0, 3) and (1, 4).
- After the paste operation, colors of the pixels at (0,3), (1,3) and (1,4) has changed. So, we print 3 after printing the modified image!

- Input and output images are visualized in Figure 3.

Hints

- Finding orientations of the input pixels with respect to each other may help. What are the coordinates of the top left corner of the area? How about top right, bottom left and bottom right?

Part 3 (40 points)

In this part you will copy and paste an area, similar to the second part. However, before pasting it you are going to rotate the area based on the rotation instructions you are given.

Specifications

1. The first input you will be given is the image. Image format is the same as in part 1 and part 2 (25x25 with values between [0, 7] for each pixel).

2. Second input you will be given is the character `R`. Denoting the rotate operation.

3. Third and fourth inputs are the rotation instructions: direction and degree respectively. Direction is a single character `L` (counterclockwise) or `R` (clockwise) and degree is one of the `[0, 90, 180, 270]`.

4. Next two inputs are the coordinates of the corners that define the **square** area you will copy from and the area you will paste onto after rotation. Unlike in part 2, it is guaranteed that the first pixel is the top left and the second pixel is the bottom right corner of the copy area. Similarly third pixel is the top left and the fourth pixel is the bottom right corner of the paste area. The format is given below.
   Input format:
   `<copy_top_left_row> <copy_top_left_col> <copy_bottom_right_row> <copy_bottom_right_col>`
   `<paste_top_left_row> <paste_top_left_col> <paste_bottom_right_row> <paste_bottom_right_col>`
   Input example:
   ```
   0 0 2 2
   3 3 5 5
   ```

5. Copied and rotated area will fit perfectly into the given paste area.

6. The pixels, whose coordinates you are given as input, are included in copy-rotate-paste operations.

7. Copy and paste area will not overlap and they will be inside the image completely.

8. After the operation is complete, you will print the modified image and the number of pixels affected by the paste operation (i.e. the color has changed). Same format as in part 2.

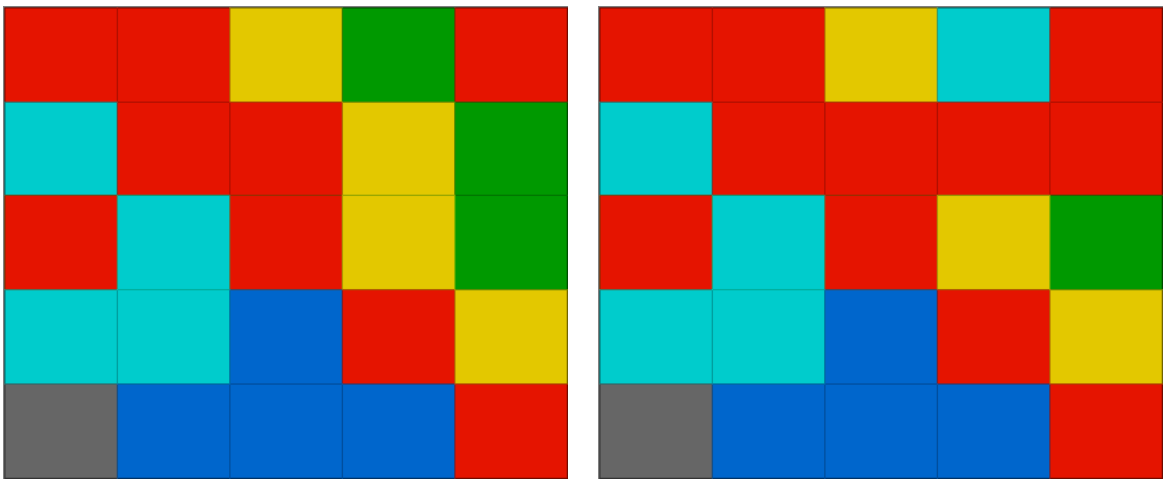
Another example and explanation!

Input:

```
0 0 1 2 0
7 0 0 1 2
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
R
R 90
0 0 1 1
0 3 1 4
```

Output:

```
0 0 1 7 0
7 0 0 0 0
0 7 0 1 2
7 7 5 0 1
6 5 5 5 0
3
```



(a) Before copy-rotate-paste operation (b) After copy-rotate-paste operation

Figure 4: Before and after the copy-rotate-paste operation.

Explanation:

- First `R` says that the operation is going to be copy-rotate-paste.
- Third input `R 90` gives us the rotation direction right (or clockwise) and the degree (90 degrees).
- Input `0 0 1 1` gives us the two pixels at (0, 0) and (1, 1) and these two pixels together define the area with the following pixels: (0,0), (0, 1), (1, 0), (1, 1). We know which

corner each pixel corresponds to (first pixel is top left and second pixel is bottom right) and the area will always be a square. So our job is easier this time.

- Similarly the area we will paste onto are defined by the pixels (0, 3) and (1, 4) and always a square.
- Three pixels are affected by this operation so we print 3 after printing the modified image!
- Input and output images are visualized in Figure 4.

Hints

- For figuring out rotation formulas you can use rotation matrices.

Regulations

1. **Programming Language:** C
2. You are not allowed to use any dynamic memory allocation.
3. Your source files will be compiled as follows:
   `gcc e1234567_the1.c -Wall -ansi -pedantic-errors -o e1234567_the1`
4. **Cheating:** We have zero tolerance policy for cheating. Your solution must be your own. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. Sharing code between students or using third party code is strictly forbidden. Even if you take only a "part" of the code from somewhere or somebody else, this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar.
5. **Newsgroup:** You must follow COW **daily** for discussions, possible updates, and corrections.
6. **Submission:** Submission will be made via CengClass. Upload a single file named e[ID]_the1.c where [ID] is your 7-digit student ID (example: e1234567_the1.c).
7. **Evaluation:** Your codes will be inspected manually to verify that you solved the first part recursively and the second part iteratively. After that your outputs will be compared to the correct outputs using black-box technique. So, make sure that you follow the output guidelines.