Lab 7: Arrays and Pointers
Part 2.

Due date: Monday, November 17, 11:59pm.

Lab Assignment

Assignment Preparation

Lab duration. Due date for **all assignments** in Lab 7 is now **November 17** (Monday).

Lab type. This is an **individual lab**. Each student will submit his/her set of deliverables.

Collaboration. Students are allowed to consult their peers[1] in completing the lab. Any other collaboration activities will violate the *non-collaboration* agreement. No direct sharing of code is allowed.

Purpose. You will implement a number of programs which use one- and two-dimensional arrays and employ functions with out parameters.

Programming Style. All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties.

Testing and Submissions. Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

[1] A peer for the purpose of CPE 101 is defined as *"student taking the same section of CPE 101"*.

compiler settings will receive an automatic score of 0.

Program Outputs must co-incide. Any deviation in the output is subject to penalties (e.g., use of different words in the output).

Please, make sure you test all your programs prior to submission!

The Task

Note: Please consult the instructor if any of the tasks are unclear.

You will use multidimensional arrays to create a library of functions that draw simple graphical images.

Using arrays in the assignment

Up until now all your programs that created PPM files did so *on-the-fly*, i.e., by computing the color of individual pixels and immediately outputing it. This approach makes it hard to draw complex images, where multiple objects exist and may potentially overlap each other.

In this lab you will use a different approach to creating graphical images. You will store the image that is being created in a *multi-dimensional array*. You will develop a library of function that draw specific shapes (circles, rectangles) by changing the contents of the array representing the image.

PPM images as arrays. A PPM image allows us to specify the color of each pixel using three components: Red, Green and Blue (RGB). Each component represents the intensity of the appropriate color. As such, a PPM image can be viewed as a three-dimensional array of color intensities. The first two dimensions, **width** and **height** represent the individual pixel. Each pixel itself is an array of *three color intensities*.

The array representing the image can be, thus declared as follows:

```
#define WIDTH 600
#define HEIGHT 400
#define COLORS 3

...

char image[HEIGHT][WIDTH][COLORS]
```

When `image` is defined this way, `image[25][56][1]` refers to the green (second) color intensity of the pixel with coordinates (56, 25) (column 56, row 25 from the top). Recall that coordinates of pixels will start at (0,0), representing the top left corner of the image. The last element of the `image` array is `image[399][599][2]`, representing the blue component of the pixel in the bottom right corner of the image.

Colors as arrays. You will be developing a library of functions which take as input the image array (it will be used as an out parameter, i.e., your functions will be modifying the contents ofthe array), the coordinates necessary to correctly place the object on the image, and the color of the object. To keep the number of function paramenters down, all colors in your library will be represented as arrays of three components. E.g., the `.c` files used to test your library, declare the variable used to represent the color of an object as

```
char color[COLORS];
```

(remember, `COLORS` was `#define`d as 3). `color[0]` will represent the intensity of the **R**ed component, `color[1]` — the intensity of the **G**reen component, and `color[2]` — the intensity of the **B**lue component.

All functions that "draw" objects on the image will take as one of their inputs an array such as the above.

Using multidimensional arrays as function parameters. Remember that **all arrays passed to functions** are treated as **out parameters**, i.e., the values stored in the elements of the arrays can be changed from inside the functions. This is true for *multidimensional* arrays as well. Multidimensional array parameters are declared in function declarations as follows;

```
int drawImage(char image[][WIDTH][COLORS]);
```

Note, that the size of the first array dimension (image height, in our case0 need not be included in the array declration – in exactly the same manner, as we did not need to declare the size of a one-dimensional array, when passing it to a function. However, the sizes of **all subsequent dimensions** have to be explicitly specified in the function declaration.

Function calls that *passs* arrays to the functions are straightforward: you simply put the name of the array variable. For example,all test files declare a `char image[HEIGHT][WIDTH][COLORS]` array to store an image, and end with the following function call;

```
  drawImage(image);
```

The geometric objects library: `image.h`

You shall create a `.h` library that contains functions manipulating an array representing a PPM image. Name your library file `image.h`. Note, that for this assignment, this is the only file you will be submitting. To test the work of your library, I am providing a number of C pograms, that use the functions from the `image.h` library to create some PPM files.

Your library shall include the following functions:

```
void drawImage(char image[][WIDTH][COLORS]);
void printHeader(int w, int h);
void setColor(char color[], char r, char g, char b);
void blankImage(char image[][WIDTH][COLORS], char color[]);
void putPixel(char image[][WIDTH][COLORS], int i, int j, char color[]);
void putRectangle(char image[][WIDTH][COLORS], int topY, int topX,
                   int height, int width, char color[]);
void putCircle(char image[][WIDTH][COLORS], int centerY, int centerX,
                   int radius, char color[]);
void putRing(char image[][WIDTH][COLORS], int centerY, int centerX,
```

```
                   int radius1, int radius2, char color[]);
void putLine(char image[][WIDTH][COLORS],
             int fromY, int fromX, int toY, int toX,
     char color[]);
```

`void drawImage(char image[][WIDTH][COLORS]).` This function takes as input the image array, and outputs to *standard otput* the PPM image file representing the image in the array.

`void printHeader(int w, int h).` This function takes as input two numbers specifying the size of a PPM image, and prints to the standard output the first three lines of the binary PPM image, i.e., the magic number (`"P6"`), the width and height of the image, and the range of color intensity values (255).

`void setColor(char color[], char r, char g, char b).` This function takes as input the "color" array and three color intensities for the red (`char r`), green (`char green`) and blue (`char b`) components. The function shall assign the appropriate color intensity values to the elements of the `color` array.

This function mostly exists for your (and my) convenience. In the C programs that use the library, I can now write statements of the sort:

```
char black[COLORS], red[COLORS], yellow[COLORS];

setColor(black, 0,0,0);
setColor(red, 255, 0, 0);
setColor(yellow, 255,255,0);
```

The color arrays can then be used when calling the remaining functions from the library.

`void blankImage(char image[][WIDTH][COLORS], char color[]).` This function takes as input the image array and the color array. It sets the color of each pixel in the image array to be the color represented by the `color` array.

For example, the following C program:

```
#include "image.h"
int main() {
   char image[HEIGHT][WIDTH][COLORS];
   char color[COLORS];

   setColor(color, 255,0,0);
   blankImage(image,color);
   drawImage(image);
   return 0;
}
```

outputs an all-red PPM image.

`void putPixel(char image[][WIDTH][COLORS], int i, int j, char color[])` This function takes as input the following parameters:

`char image[][WIDTH][COLORS]`	image array
`int i, int j`	the row and the column of a pixel
`char color[]`	color array

It sets the color of pixel `(i,j)` in the `image` array to be the one represented by the `color` array.

For example, the following C program:

```
#include "image.h"
int main() {
   char image[HEIGHT][WIDTH][COLORS];
   char black[COLORS], color[COLORS];

   setColor(black, 0,0,0);
   setColor(color, 255,0,0);
   blankImage(image,black);
   putPixel(image, 200,300,color);
   drawImage(image);
   return 0;
}
```

outputs a PPM image of a single red pixel at coordinates (200,300) on the all-black background.

`void putRectangle(char image[][WIDTH][COLORS], int topY, int topX, int height, int width, char color[]).` This function takes the following parameters:

`char image[][WIDTH][COLORS]`	image array
`int topY, int topX`	the top left corner of the rectangle
`int height, int width`	the height and the width of the rectangle
`char color[]`	color of the rectangle

This function changes the pixels of the `image` array to show a rectangle with top right corner at `(topX,topY)` (row `topY`, column `topX`), of witdth `width` and height `height` using the color specified by the `color` array.

For example, the following program:

```
#include "image.h"
int main() {
   char image[HEIGHT][WIDTH][COLORS];
   char black[COLORS], color[COLORS];

   setColor(black, 0,0,0);
   setColor(color, 255,0,0);
   blankImage(image,black);
   putRectangle(image, 50, 50, 200,300,color);
   drawImage(image);
   return 0;
}
```

outputs a PPM image showing a red rectangle on black background.

`void putCircle(char image[][WIDTH][COLORS], int centerY, int centerX, int radius, char color[]).` This function takes as input the following parameters:

`char image[][WIDTH][COLORS]`	image array
`int centerY, int centerX`	center of the circle
`int radius`	radius of the circle
`char color[]`	color of the circle

This function changes the pixels of the `image` array to show a solid circle of radius `radius` centered at `(centerX, centerY)` (column `centerX`, row `centerY`). using the color specified by the `color` array.

For example, the following program:

```
#include "image.h"
int main() {
   char image[HEIGHT][WIDTH][COLORS];
   char white[COLORS], color[COLORS];

   setColor(white, 255,255,255);
   setColor(color, 255,0,0);
   blankImage(image,white);
   putCircle(image, 200,300,100, color);
   drawImage(image);
   return 0;
}
```

outputs a PPM image similar to the flag of Japan (a red circle in the center of a white field).

`void putRing(char image[][WIDTH][COLORS], int centerY, int centerX, int radius1, int radius2, char color[]).` This function takes as input the following parameters:

`char image[][WIDTH][COLORS]`	image array
`int centerY, int centerX`	center of the ring
`int radius1`	outer radius of the ring
`int radius2`	inner (smaller) radius of the ring
`char color[]`	color of the ring

This function changes the pixels of the `image` array to show a ring with outer radius `radius1` and inner radius `radius2` centered at `(centerX, centerY)` (column `centerX`, row `centerY`). using the color specified by the `color` array.

For example, the following program:

```
#include "image.h"
int main() {
   char image[HEIGHT][WIDTH][COLORS];
   char white[COLORS], color[COLORS];

   setColor(white, 255,255,255);
   setColor(color, 255,0,0);
   blankImage(image,white);
   putRing(image, 200,300,100, 60, color);
   drawImage(image);
```

```
    return 0;
}
```

outputs a PPM image with a red ring of width 40 (100 - 60) on the white background.

`void putLine(char image[][WIDTH][COLORS], int fromY, int fromX, int toY, int toX, char color[])`. **This is an extra credit assignment.** If you do not want to complete it, or, if you were unable to get it right, **include in your `image.h` file a stub of this function**, i.e., a function that does nothing (doing so, allows me to run all tests, including extra credit tests on all submissions).

This function takes as input the following parameters:

`char image[][WIDTH][COLORS]`	image array
`int fromY, int fromX`	the starting point of the line
`int toY, int toX`	the ending point of the line
`char color[]`	color of the line

The function paints pixels of the `image` array to show a line of color specified by the `color` array, connecting pixels `(fromX,fromY)` and `(toX,toY)`.

When implementing this function, please note the following:

- Any pair of pixels can be connected by a line. The choice of the start and the end pixel of the line is arbitrary. That is, given two pixels: `(y1,x1)` and `(y2,x2)`, `putLine()` can be called in two different ways:

  ```
  putLine(image,y1,x1,y2,x2,color);
  ```

  and

  ```
  putLine(image,y2,x2,y1,x1,color);
  ```

  to produce the line between them.

- Any line on a pixelated is an approximation of a real straight line between two points in space. Thus, lines may and will look pixelated. It may be a good idea to make the line at least two pixels wide to make it appear smoother.

Submission.

Files to submit. For this part of the lab you shall submit one file: `image.h`.

Submission procedure. Use`handin` program to submit your work. Ssh to `vogon` (it may be possible now to submit without having to ssh to `vogon` — feel free to try it), and enter one of the two commands:

Section 09:

```
> handin dekhtyar lab07-09 image.h
```

Section 11

```
> handin dekhtyar lab07-11 image.h
```

Appendix A. Testing

For this part of the lab, testing is done somewhat differently. Previously, your test cases consisted of a number of text files that were fed as inputs to an executable file compiled from the `.c` program you submitted.

For this assignment you do not submit a `.c` program. Rather, you submit a `.h` header file containing a library of functions. To test it, you are given a collection of `C` programs that `#include` your `.h` file, and use functions defined in it.

The collection of test programs is divided into two groups:

1. Unit tests. These are simple programs that test a specific library function. Examples of unit test programs are given in this text. Note, that all programs call `blankImage()` and `drawImage()` (and, thus, `printHeader()`).

2. Advanced tests. These programs are designed to test all components of your library working together. They also show you that even with the simple image-drawing tools that you have, it is possible to draw some rather sophisticated pictures.

In addition to the C programs, I am also providing the outputs for each program in the PPM format. Finally, a test script that compiles and runs all tests is also provided.

Note: to make test script run smoothly, start development of `image.h` header file by creating **stubs** of all functions. Note, a stub of a function is a function definition that does nothing, and (if necessary) produces token (dummy) output. In your case, the body of a function stub need only include one statement: `return;`.

Having all functions implemented as stubs will allow you to compile and run all test programs at any moment.

Note: Test programs output the PPM file to standard output, so they should be run using output redirection. **Always check the size of the files created by your program**. For this assignment, you are working on files of size 600×400, so the size of the file should be around 720Kb (a bit less, actually). If your output PPM file size is greater, your code (most likely, `drawImage() function` contains bugs.