Lab 5: Loops and Functions...

Due date: Thursday, October 23, 11:59pm.

Lab Assignment

Assignment Preparation

Lab type. This is a **pair programming lab**. For this lab, your partner is assigned to you randomly. Partner assignment will occur at the beginning of the lab period on Tuesday. Your partner **will be different** from your Lab 4 partner. See the pair programming handout from the first day of classes for more information on what is expected of you.

Collaboration. Students work in pairs, and it is considered cheating, if members of the team (pair) do not work together. Communication between pairs during lab time is allowed, but no direct sharing of code is allowed.

Purpose. The lab allows you to practice the use of functions, conditional statements and loops.

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. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies **to each assignment** (lab, programming assignment, homework) in this course.

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

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

compiler settings will receive an automatic score of 0.

For each program you have to write, you will be provided either with instructor's executable and a battery of tests or with the output of the instructor's program. The programs you submit must pass all tests made available to you and/or the output must co-incide with the instructor's output.

```
> japan > japan.ppm
> diff alex-japan.ppm japan.ppm
```

Program Outputs must co-incide. Any deviation in the output is subject to penalties. **PLEASE, USE BINARY EXECUTABLES PROVIDED BY THE INSTRUCTOR!**. The exception is made in case of floating point computations leading to differences in the last few decimal digits. You can check whether or not a program produces correct output by running the `diff` command:

Please, make sure you test all your programs prior to submission! Feel free to test your programs on test cases you have invented on your own.

The Task

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

In this lab you, working in pairs, will perform the following tasks:

1. Write a few programs that create PPM images of varying complexity.
2. Improve your currency converter program.

Task 1: PPM Files

Note: The **PPM format explanation** is now in the Appendix B.

General Instructions

1. All programs that output ppm images should use stdout. We will be creating files using output redirection.
2. For this assignment, use exactly the colors specified in the color table found in Appendix B.
3. All images you are asked to produce are also available from the course web page.
4. For some of the images, you are given instructions on **how** your program should be implemented. **Your code will be graded for conformance to these instructions.**

Image 1: Flag of Japan: japan.c

Your first image is a flag of Japan. The flag of Japan is a white field, with a red circle in the middle.

Your program shall output the .ppm file representing the national flag Japan. Name your program `japan.c`.

Image dimenstions: 600×400
Circle center: (300,200)
Circle radius: 120

Instructions. Many image drawing programs can be implemented in the following manner:

- `main()` function contains a double-nested `for` loop that iterates over every single pixel of the image.
- For each pixel (represented by coordinates, say (x,y)), the program calls a function that determines the color of the pixel and returns it.
- The `main()` function uses a sequence of `if` statements or a `switch` statement to output the color of the pixel.

You will implement this program using the approach described above. In particular, your program shall contain the following function:

```
int inCircle(int i, int j)
```

This function takes as input the pixel coordinates i (row) and j (column), and outputs 1 if the pixel is inside the red circle (i.e., has to be colored red) and 0 if the pixel is outside of the red circle (i.e., has to be colored white).

Image 2: US Flag without stars: us-flag.c

The next image you will create is an (almost) US flag.

The US flag consists of 13 red and white stripes, and a dark blue field at the top left corner. The top and bottom stripes are red, so, the flag has 7 red and 6 white stripes. The real flag contains 50 stars on the blue field, but the image you are creating will not have start (this will be done at a later lab). The height of the blue field matches the height of seven stripes.

Name your program `us-flag.c`.

The parameters of the image are:

Image dimenstions:	780×520
Height of a stripe:	40
Blue field height:	280
Blue field width :	360

Instructions. There are four types of rows on the flag:

- blue field – red stripe
- blue field – white stripe
- red stripe
- white stripe.

Use four `paint` functions. Each function is repsonsible for outputing to stdout one row of the image, corresponding to one of the four types above.

The functions are:

```
int paintBlueRed();
int paintBlueWhite();
int paintRed();
int paintWhite();
```

The functions accept no parameters, and shall always return 0. (your `main()` function does not need to capture the return values of these functions).

The `main()` program shall use the four functions above to create the output.

Note: Use dark blue color from Appendix B for the blue field of the flag.

Image 3: Flag of Greece: greece.c

The third image you will create is a flag of Greece. The flag consists nine alternating blue and white horizontal stripes. The top and the bottom stripe are blue. The top left corner of the flag contains a square blue field with a white Greek cross (a cross whose sides are equal) centered inside it. The height/width of the blue field is equal to the height of five stripes. The width of the cross sides is equal to the width of the blue and white stripes on the flag.

Write a C program that outputs the flag of Greece. Name it `greece.c`. The parameters are:

Image dimensions:	540×360
Height of a stripe:	40
Height/Width of the blue field:	200 (= 5 stripes)
Cross:	centered inside the blue field

Instructions. Your program shall contain no functions but the `main()` function shall contain a *meaningful* **triple-nested loop**.

Image 4: Diagonals: diagonals.c

You will create a square image. The image shall contain two diagonals which bissect the image into four triangles, which we will refer to as North, South, East and West. Each triangle has a different color.

Image Feature	Color
North triangle	blue
East triangle	red
South triangle	yellow
West triangle	green
diagonals	black

Name your program diagonals.c. Some other parameters for the program:

Image size:	800×800
Width of diagonal line:	1 pixel

Instructions. Use four functions to establish if a pixel belongs to a specific triangle.

```
int checkNorth(int i, int j);
int checkEast(int i, int j);
int checkSouth(int i, int j);
int checkWest(int i, int j);
```

Each of the functions returns 1 if the pixel *(i,j)* belongs to the appropriate triangle, and 0 - otherwise.

The `main()` function shall use the `check` function to determine which triangle the pixel belongs to (note, if the pixel does not belong to any triangle, it should mean it is on the diagonal), and paint the pixel according to its position.

Image 5: Simple gradient: gradient.c

Up until now, we have been constructing images that used pre-defined colors to color specific, well-established regions. This program asks you to create an image that will color each pixel into a different color.

The gradual change of color from pixel to pixel is usually referred to as *gradient* effect. You will create a program that outputs a PPM image illustrating this effect. Your program will output a square image with the following parameters:

Image size:	256×256
Color: top left corner:	black
Color: top right corner:	blue
Color: bottom left corner:	red
Color: bottom right corner:	purple
Horizontal gradient:	black to blue
Vertical gradient:	black to red

Name your program `gradient.c`.

Instructions. Basically, you need to figure out how to assign a color to each pixel.

Task 2: New currency exchance program

You are now ready to complete the currency exchange program.

The new version of the currency exchange program shall be named `exchange.c`. The program shall perform the following actions.

EX-1. The output of the program shall coincide, character for character (and space for space) with the output of the instructor's program. The only discrepancies are allowed in the low decimals of the floating point numbers returned by the program.

EX-2. Upon startup, the program shall output the text describing its purpose (see sample output for the exact format), and request the exchange rate. **Note:** the exchange rate is **the number of Russian rubles needed to purchase $1**. Your program shall ensure that a *positive* number is provided. After that, the program shall request the commission on the exchange. The program shall ensure that the commission entered is non-negative and is less than 100.

EX-3. The program shall contain a main action cycle. The main cycle consists of three major parts:

1. Main Menu. The program provides a menu of choices to the user.
2. Transaction: input. The program processes user's choice from the menu, asks for further input if needed.
3. Transaction: output. The program completes the transaction.

EX-4. The main menu of the program consists of three choices:

1. Exchange USD to RR.
2. Exchange RR to USD.
3. Quit the program.

At the beginning of each cycle, your program shall output the menu, and provide a prompt requesting user choice. Sample output shows the way the main menu is arranged on screen as well as the text of the prompt.

EX-5. The program shall read user choice. Valid values of the user choice are $1, 2, 3$, all other values are invalid. The program shall repeat display of the main menu and the prompt, until a valid choice is entered.

EX-6. Quit. If the user enters 3, the user has selected to quit the program. Your program shall display a good-bye message (see sample output) and quit.

EX-7. USD to RR. If the user enters 1, the user has selected to exchange US dollars into Russian Rubles. Your program shall prompt the user for the number of US dollars, that the user wants to exchange (the number must be an integer, the exchange kiosk does not accept deals that use fractions of dollars). The amount entered must be positive, your program shall enforce that.

After the amount is entered, your program shall compute the amount of Russian rubles to be returned to the user and the amount of commission (in rubles). These two numbers shall be returned to the user: see sample output for details. **Note:** your output must break the amounts into rubles and copecks.

EX-8. RR to USD. If the user enters 2, the user has selected to exchange Russian Rubles for US Dollars. Your program shall prompt the user for the number of Russian rubles, that the user wants to exchange (the number must be an integer, the exchange kiosk does not accept deals that use fractions of rubles). The amount entered must be positive, your program shall enforce that.

After the amount is entered, your program shall compute the amount of US dollars to be returned to the user and the amount of commission (in rubles). These two numbers shall be returned to the user: see sample output for details. **Note:** your output must break the amounts into dollars and cents, and rubles and copecks for the exchange amount and the commission respectively.

EX-9. After the transaction is completed, the program returns to the state describe by **EX-4**.

Instructions

Use of functions. You must use at least three different functions in your program. One function, `round()` is described below. The other two functions are left up to you (let it be a design exercise for you), but they have to be **meaningful** (i.e., they have to perform a well-defined, structured task).

Function `round()`. To compute correctly the copecks and the cents in the exchanges you need a proper rounding function. Unfortunately, C standard library contains only `floor()` (which you also may need to use for this program) and `ceil()`, but not a proper rounding function. You shall declare and define the rounding function `int round(float x)` in your program. My rounding function rounds 5.5 to 6 but 5.49999 to 5. Yours should do the same.

Sample Output

Session 1: no fractions.

```
> gcc -ansi -Wall -Werror -lm -o exchange exchange.c
> exchange
Currency Converter Kiosk: USD to RR

Enter exchange rate: 10
Enter commission%: 5
- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?1
How much money do you want to change? 100

Here are your 950 rubles 0 copecks
The commission was 50 rubles 0 copecks


- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?2
How much money do you want to change? 10000

Here are your 950 dollars 0 cents
The commission was 500 rubles 0 copecks


- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?3
Thank you for using our services.
```

Session 2: Fractions.

```
> exchange
Currency Converter Kiosk: USD to RR

Enter exchange rate: 10.50
```

```
Enter commission%: 1
- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?1
How much money do you want to change? 100

Here are your 1039 rubles 50 copecks
The commission was 10 rubles 50 copecks


- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?2
How much money do you want to change? 5000

Here are your 471 dollars 43 cents
The commission was 50 rubles 0 copecks


- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?2
How much money do you want to change? 2343

Here are your 220 dollars 91 cents
The commission was 23 rubles 43 copecks


- - Menu - -
1. USD --> RR     2. RR --> USD      3. Quit

What would you like to do?3
Thank you for using our services.
```

Submission.

Files to submit. You shall submit **seven** files:

> `team.txt`, `japan.c`,`us-flag.c`,
> `greece.c`, `diagonals.c`, `gradient.c`, `exchange.c`

`team.txt` file shall contain the name of the team and the names of the two team members in each pair, and the Cal Poly IDs of each. E.g, if I were on the team with Dr. John Bellardo, my `team.txt` file would be

```
Go, Poly!
John Bellardo, bellardo
Alex Dekhtyar, dekhtyar
```

No other files can be submitted. In fact, if you submit *other* file, or submit one of the three files about with an *incorrect filename*, you will receive an email informing you about a submission error, and asking you to resubmit.

Files can be submitted one-by-one, or all-at-once.

Submission procedure. You will be using `handin` program to submit your work. The procedure is as follows:

- `ssh` to `vogon` (`vogon.csc.calpoly.edu`).
- Students from **Section 009** shall execute the following submission command:

```
> handin dekhtyar-grader lab05-09  <your files go here>
```

- Students from **Section 011** shall execute the command:

```
> handin dekhtyar-grader lab05-11 <your files go here>
```

`handin` is set to stop accepting submissions 24 hours after the due time.

Grading

Any submitted program that does not compile earns 0 points.

Any submitted program that compiles but fails at least one **public** (i.e., made available to you) test (or produces visibly incorrect output, or fails the `diff` test) earns no more than 30% of its full score (and can possibly earn less).

Any submitted program that compiles and succeeds on **all** publically available tests earns at least 50% of its full score. The PPM creation programs that succeed both the visual inspection and the `diff` test earn 100%.

All programs will be checked for style conformance. Any style violation will be noted. The program will receieve a 10% penalty.

Appendix A. Testing

Instructor's executables. Instructor's executable for the `exchange` program is available on the course web page. Instructor's PPM images for all other programs are also available. When copying executable files, please make sure to run `chmod u+x` on them.

Public test suites. Public test suit for the `exchange` program is available on the web page.

Appendix B: PPM Format Explanation

Portable Pixel Map (`.ppm`) file format is a simple format for storing graphical images. Files in this format can easily be created using C programs.

Basics. An computer image is a two-dimensional grid of pixels. Each pixel represents the smallest undivisible part of the computer screen. An image file is an assignment of color to each pixel. Pixels are referred to by their Cartesian coordinates. The top left corner of an image has the coordinates (0,0). The bottom right corner has the coordinates (n, m) where n is the width of the image in pixels and m is the height of the image in pixels.

Colors. Portable pixel map files use RGB (Red, Green, Blue) color format to represent the color of each pixel. In RGB format, a color of a single pixel is separated into three components: the red component, the green component and the blue component. The final color of an RGB pixel is determined by combining the Red, Green and Blue components into a single color.

In our course, all individual RGB component intensities range from 0 (not visible) to 255 (highest intensity) and are represented as integer numbers. RGB color (0,0,0) is black, RGB color (255,255,255) is white. The table below contains the list of colors used in this lab and their RGB values.

Color	RGB Red	RGB Green	RGB Blue
black	0	0	0
white	255	255	255
red	255	0	0
green	0	255	0
blue	0	0	255
yellow	255	255	0
purple	255	0	255
orange	255	128	0
dark blue	0	0	80

File format. There are two PPM formats: a "raw" PPM file and a "plain" (ASCII) PPM file. ASCII PPMs are human-readable, but they take too much space. Raw PPMs are smaller in size, but cannot be read by a human. In this lab you will be generating raw PPM files.

From http://netpbm.sourceforge.net/doc/ppm.html (with some modifications):

Each PPM image consists of the following:

1. *A "magic number" for identifying the file type. A ppm image's magic number is the two characters* `"P6"`.
2. *Whitespace (blanks, TABs, CRs, LFs).*
3. *A width, formatted as ASCII characters in decimal.*
4. *Whitespace.*
5. *A height, again in ASCII decimal.*
6. *Whitespace.*
7. *The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.*
8. *A single whitespace character (usually a newline).*
9. *A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of green, blue and red intensities, in that order*[1].

Representing Colors in C. Outputing raw PPM files is actually quite simple. The idea is to use `unsigned char` variables to store information about RGB intensities.

[1] This is what worked for me. If your colors don't look right, switch to RGB order.

Variables of type `char` and `unsigned char` are treated by C both as a character and as a number in the range -128 – 127 or 0 — 255 respectively. The following code outputs an RGB triple to stdout.

```
unsigned char Rcolor, Bcolor, Gcolor;
Rcolor = 255;
Bcolor = 0;
Gcolor = 128;

printf("%c%c%c", Gcolor, Bcolor, Rcolor);
```