Homework 2. . .

Due date: Tuesday, October 28

Assignment	Due	Late submission policy
Paper-and-pencil part:	beginning of the class	late submissions **not accepted**
Programming part:	11:59pm.	standard policy: 24 hour late submission period, up to 30% penalty

Submission Instructions

The assignment has two parts: (i) the programming part, extending on your Lab 4-5 work and (ii) the paper-and-pencil part, assigned in preparation for the midterm..

- The paper-and-pencil part of the assignment should be detached from this handout, completed and submitted before the beginning of the midterm on Tuesday, October 28.
- The programming part should be submitted using the following `handin` commands:
  - Section 09:
    ```
    > handin dekhtyar hw02-09 <file1>,...,<fileN>
    ```
  - Section 11:
    ```
    > handin dekhtyar hw02-09 <file1>,...,<fileN>
    ```

The list of files to submit is supplied below.

Assignment Preparation

Collaboration. This is an **individual assignment**, **but you are allowed to collaborate with your Lab 5 partner on the programming part**. Each student has to develop and submit his/her own version of each program, but Lab 5 partners can work together on problem-solving aspects of the assignment. No other form of collaboration is permitted.

Programming Style. All submitted C programs for the programming part 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.

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.

All `.ppm` image files will be tested against the instructor's `.ppm` files using `diff`. **Make sure you do so as well**.

Programming Assignment.

Files to submit. For the programming part of the assignment you shall submit **five files:** files:

Program 1: Target: target.c

Design an implement a program that outputs a `.ppm` image resembling a logo of the Target store chain. Name the program `target.c`.

The image your program is creating is a white square in the red circle in the middle, and with a red ring around it. The parameters of the program are:

Image dimenstions:	500×500
Circle center, ring center:	(300,200)
Circle radius:	50
Small ring radius	100
Ring width	20
Circle and ring color:	red

Instructions. Follow the same idea as you used for display of the flag of Japan. This time you will have to implement two functions:

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

Both functions take as input the pixel coordinates i (row) and j (column), and output 1 if the pixel is inside the red circle /red ring (i.e., has to be colored red) and 0 if the pixel is outside of the red circle/red ring respectively (i.e., has to be colored white).

Program 2: Flag of Jamaica: jamaica.c

Design an implement a program that outputs a `.ppm` image Of the flag of Jamaica. Name the program `jamaica.c`.

The flag of Jamaica consists of two triangular black fields on the left and the right side of the flag and two triangular green fields at the top and the bottom of the flag separated from each other by a yellow diagonal cross.

The parameters of the flag and the image are:

Image dimenstions:	600×300
Vertical offset for the yellow field:	25
Horizontal offset of the yellow field:	50

Note: the vertical and horizontal offsets in the table above are as follows. The horizontal offest specifies the horizontal width of each yellow line forming the cross. The vertical offset specifies its vertical witdth.

Instructions. Implement your program using two functions:

```
int checkBlack(int i, int j);
int checkYellow(int i, int j);
```

Both `checkBlack()` and `checkYellow()` take as input the coordinates of a pixel and determine if the pixel should have the designated color. `checkBlack()` returns 1 if the pixel should be black and retruns 0 otherwise. `checkYellow()` returns 1 if the pixel should be yellow, and returns 0 otherwise.

Use these functions in the `main()` function to output the image.

Program 3: Olympic Rings: rings.c

Write a program that produces a ppm image showing five Olympic rings. The image you will be constructing is a simplified version of the actual Olympic symbol. (the simplification is in how overlaps of the rings are handled).

The Olympic rings symbol is an image of five rings on the white background. The rings are arranged with three rings in the top row and two rings – in the bottom row. The rings in the same row do not intersect, but the neighboring rings from different rows are intertwined. Each ring has a different color, symbolizing one of the five contintents (Europe, Americas, Asia, Africa and Australia). In your image, the top row rings will be on the "foreground", while the bottom row rings will be on the background.

The parameters of the image are:

Image dimenstions:	900×600
Top row :	200
Bottom row: 260	
Top row X coords of ring centers:	270, 450, 630
Bottom row X coordinates of ring centers:	360, 540
Inner radius of rings:	70
Ring width:	80
Top row ring colors (left to right):	blue, black, red
Bottom row ring colors (left to right):	yellow, green

Instructions. Implement this program by writing two functions:

```
int getColor(int i, int j);
int inRing(int i, int j, int centerX, int centerY)
```

`getColor()` takes as input the coordinates of a pixel and returns a number between 0 and 5. The number determines the color of the pixel as follows:

getColor() returns:	Pixel Color
0	white
1	blue
2	black
3	red
4	yellow
5	green

`inRing(int i, int j, int centerX, int centerY)` takes as input four arguments. The first two (`i,j`) describe a pixel of the image. The second two (`centerX`, `centerY`) are the coordinates of the center of a ring. Given these arguments, `inRing()` returns 1 if the pixel belongs to the ring, with the specified center, and 0, if the pixel does not belong to the ring.

Your `main()` function should include only calls to `getColor()`. Your `getColor()` should call `inRing()` (possibly, multiple times).

Program 4: Extra credit: rising sun: rising.c

This is an **extra credit** program.

Write a program that produces the `rising-sun-alex.ppm` image (the image is made available to you from the course web page). Name the program `rising.c`.

The image is the flag of Japan, modified by careful use of gradient coloring. In particular, the red circle is colored using gradient coloring to create an illusion that it is a ball, lighted from the top left corner of the image. The red circle resides blue-green gradient background. The parameters of the image are:

Image size:	800× 400
Circle center	(400, 200)
Circle radius	100
Center of the "light" on the circle:	(360, 160)
Gradient color scale:	4

Note: no restrictions on implementation. As long as you are able construct a similar image (green-blue gradient background, reasonably similarly shaded sun on the foreground) you are eligible for partial credit for this problem.

Program 5: boring01.c

Write a program that reads from standard input a sequence of integer numbers, find the maximum number and outputs it to the user. The number of inputs is unknown to the program, but the last number in the input will always be 0.

Name your program `boring01.c`

Your program shall produce no output other than the max number.

Sample run:

```
> cat test-case
```

```
3 7 1 18 28 3 14 0
> boring01 < test-case
28
```

Program 6: boring02.c

Write the program, that reads from standard input a sequence of numbers until **one of the following conditions becomes true**:

1. The sum of all numbers read thus far exceeds (becomes strictly greater than) 100 (e.g., 10 50 60).
2. The program reads two consecutive numbers that are the same (e.g., 2 2 or 23 23).
3. The program reads two consecutive numbers that add up exactly to 25 (e.g., 13 12).

When one of these conditions becomes true, the program prints the number, reflecting the condition, which became true and quits. Name the program `boring02.c`.

Sample output.

```
> cat test1
10 20 30 10 10
>boring02 < test1
2
> cat test2
20 30 50 1
> boring02 < test2
1
>cat test3
25 5 10 14 1 24
> boring02 < test3
3
>cat test4
1 2 3 4 5 6 7 8 9 9 99 12 13
> boring02 < test4
2
```

Instructions. Create three functions testing each of the three conditions described above. Use these functions in your `main()`.

Paper-and-Pencil Part

Instructions. Detach the paper-and-pencil part, do all the work in the spaces alotted for it, and submit on the due date.

Problem 1. Expressions

Rewrite each C expression using parenthesis to show the order of operations.
(e.g. `a+b-c` is `(a+b)-c`.)

```
(a) 23 + 47 -z--                  __________________________________

(b) x >= y++*++z                  ___________________________________

(c) ++r/4+t++/3                   ___________________________________

(d) -1+--r-t++*e && o+1==4        ___________________________________

(e)  a || b && x+y == z||w&& p+=t++%--z

     ______________________________________________________________________
```

Problem 2. Assignment

Consider the following code fragment:

```
x += y - ++z;
y -= z - x++;
z *= z;
```

All variables are declared as `int`s above the fragment. For each set of initial (i.e., before the code fragment is executed) variable assignments below, specify the values of `x, y` and `z` after the code fragment executes.

```
(a) Initial:   x: 1      y: 1     z: 1

    Final:     x: ___    y: ___   z: ___


(b) Initial:   x: 20    y: 5     z: 2

    Final:     x: ___    y: ___   z: ___


(c) Initial:   x: -4     y: -5     z: 10

    Final:     x: ___    y: ___    z: ___
```

Problem 3. Comparisons and logical expressions

Consider the following assignments (all variables are `int` and declared):

```
x = 2;
y = 3;
z = x++-y++;
y += z;
```

For each expression below, specify what it evaluates to.

```
(a) x == -2                               evaluates to _________

(a) x == y                                evaluates to _________

(b) y <= z                                evaluates to _________

(c) !(z+x >= y)                           evaluates to _________

(d) y > 0 || x < 0                        evaluates to _________

(e) (z && x && y)                         evaluates to _________

(f) !(++y)                                evaluates to _________

(g) (x && (x!=z || z != y) & !(y||z))     evaluates to _________

(h) (z+z > x*y)                           evaluates to _________

(i) (z && x && !y)                        evaluates to _________

(j) (0 == 0 - (x-y-z))                    evaluates to _________
```

Problem 4. Conditionals

In the code fragments below all variables are declared as `int`.

(1) Consider the following code fragment:

```
z = 0;
if (x >= y) {
      if (x <= y+10) {
          z = 1;
      }
 }
 else {
    if (y < 100 && x*y % 2 == 0) {
       z = 5;
    }
    else {
      z++;
      ++z;
    }
  }
```

Compute the value of `z` after this code is executed, for each pair of values of `x` and `y` provided below:

```
(a) x = 5; y = 8;             z is _____________

(b) x=100; y = 10;            z is _____________

(c) y = 200; x = 300;         z is _____________
```

(1) Consider the following code fragment:

```
switch(x+y%z) {
    case 1: {break;}
    case 2: {x = 3;}
    case 3: {y = 3;}
    case 4: {x = y; break;}
    case 5: {y = x;}
    default:{x = 0;}
 }
```

Compute the values of `x` and `y` after this code is executed, for each pair of initial values of `x` and `y` and `z` provided below:

```
(a) x = 5; y = 8; z=10;        x is ______     y is ______

(b) x= 5; y = 7; z=8;          x is ______     y is ______

(c) x = 2; y = 17; z= 9;       x is ______     y is ______

(d) x= 7; y = 5; z=8;          x is ______     y is ______

(e) x = 10; y = 4; z= 13;      x is ______     y is ______
```

Problem 5. Loops.

(1) Rewrite the following `for` loop as a `while` loop:

```
for (j=2; j<= t++; j+=3) {
    printf("%d\n",j);
    j = j-1;
}
```

(2) Consider the following code fragment:

```
while (z > x+y) {
   x = (x+y)/2;
   y += x;
}
```

For each triple of values `x`, `y`, `z` determine, how many iterations the loop will have.

```
(a) x = 1; y = 1; z = 10;            # iterations: _________

(b) x = 1; y = 2; z = 11;            # iterations: _________

(c) x = 5; y = 1; z = 15;            # iterations: _________
```