CSC220 Lab 1 – Java Review

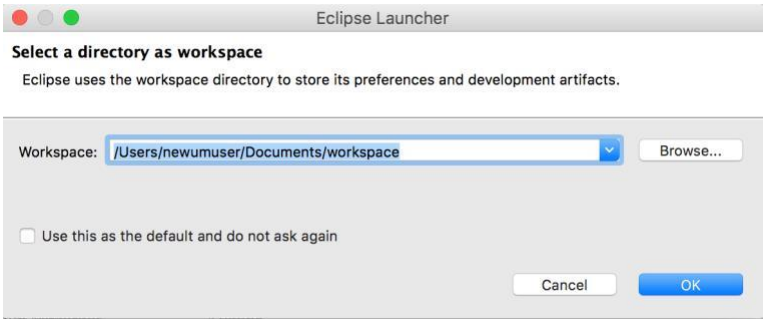
The goal of today's lab and this week's assignment is:

1. Introduce you to Eclipse
2. Give you a chance to try review Java
3. Practice uploading to your box account

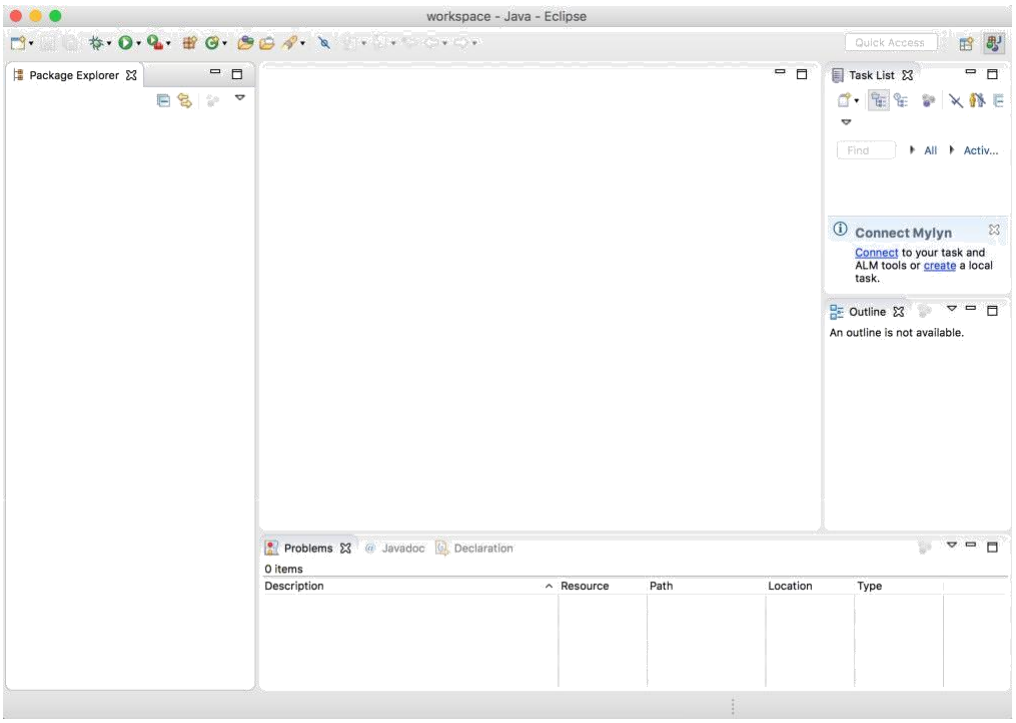
Part 1 - Create a project for CSC220 in Eclipse

We will now learn how to create a new Java project in Eclipse:

1. Launch Eclipse (consult your lab instructor to see how to launch Eclipse). All the computers in the lab have Eclipse installed, so you don't need to install it again. (If you're using your laptop, you'll need to downloadit first)
2. The very first window you will see is the following window:



3. This window is asking you where your "workspace" folder is. That means, where all your code and projects are going to be saved. Please do NOT change this path AND make a note of this path. After hitting "OK" in this page, you will be directed to the a "welcome page".
4. You are free to explore this page after the lab but none of the information in this page is required for CSC220. Simply close it by hitting the "X" sign on the top left (don't close the whole Eclipse).
5. Now you are in the main Eclipse page, where you will spend a lot of time this semester for doing your assignments ☺ and it looks like the following figure.



6. The above window consists of several large sections for managing your Java projects.
    a. **Package Explorer:** This tab, on the left, is where you will eventually see a list of your java files.
    b. **Outline:** This tab, on the right, is where you will see the organization of each Java file.
    c. **Center area:** This area is where you will edit your code.
7. Now we want to create a new Java project. We do so by selecting **File->New->Java Project** from the menu bar at the top. For the project name, enter **Lab01** **(nothing else!)** and hit **finish** to create your first Java project in Eclipse.
8. Your new project should show up in the **Package explorer** tab on the left. Open it by double clicking (or by clicking the small triangle next to the name). You will see a **src** folder and a **JRE System Library** folder. All of your work will be created in the **src** folder.
9. Right click on the **src** folder and select **New->Package**. Give the package the name **lab01** (all lower-case and no space - **nothing else!**) and click **finish**.

Now let's start programming ☺

Part 2 – A Simple Experiment

Assume you have a coin that is slightly out of balance. When this coin is flipped, it comes up heads 50.5% of the time. You think to yourself that you can get rich using this coin, so you bet with friends on the outcome of a coin flip. The winner of each coin flip gets $1 from the loser

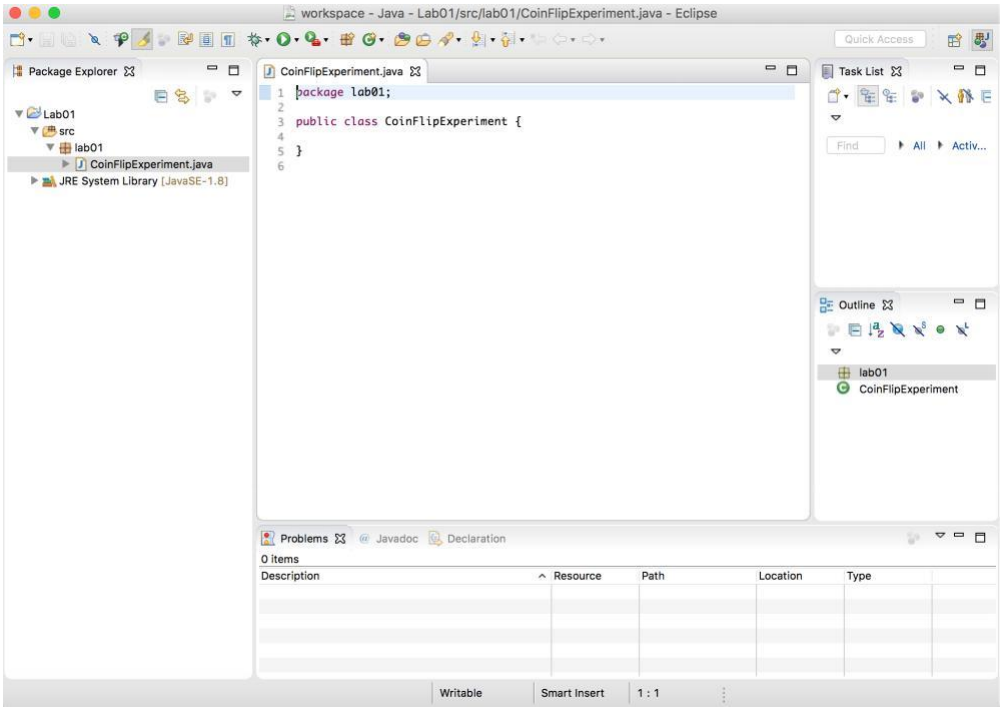
and you always bet on “heads”. Would you be very likely to win lots of money?! Let’s code it up to see…

This week’s goal is to write a Java program that does the following:

Flip the unbalanced coin 100 times, keeping track of how much you win or lose

To start, create a new class named “CoinFlipExperiment” in the “Lab01” package as follows:

1. Right click on “lab01” package that you just created. Then, select **New->Class**. In the “name” tab, type “CoinFlipExperiment” (nothing else please), and hit finish. Now you see a new file has been created under “lab01” package called “CoinFlipExperiment.java” and you see the corresponding code in the center area of the window.

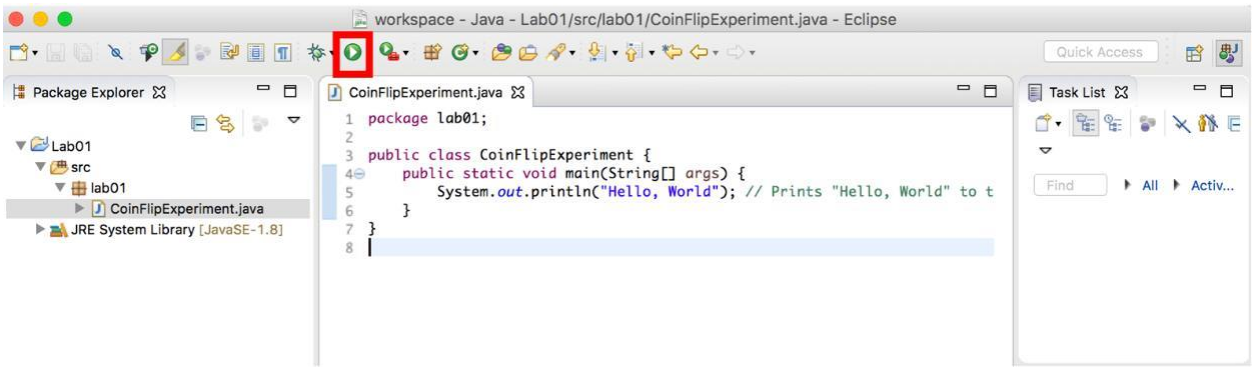


Your window should look like the figure above. Please double check the package Explorer view and the center area and if your window looks different consult with your lab instructor.

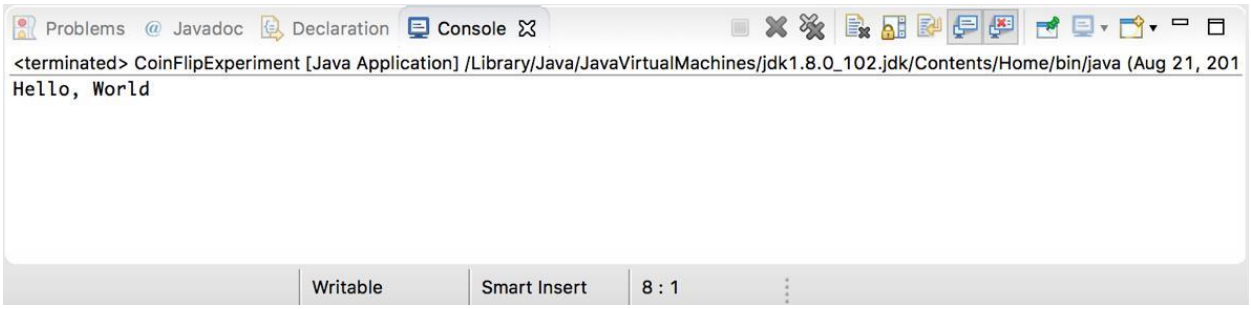
2. Now, let’s test a simple “HelloWorld” program to learn how to compile and run Java programs in Eclipse. Copy and paste the following piece of code on line 4. (If you have difficulty copying just grab a copy of CoinFlipExperiment.java available in part2 folder and place it in your workspace folder. That is: `workspace/Lab01/src/lab01`)

```
public static void main(String[] args) {
    System.out.println("Hello, World"); // Prints to the terminal window.
}
```

3. Now you are ready to run your first program in Eclipse. To run your program, click on the little green button highlighted in the picture below



4. And then you should see your "hello world" appear on the bottom portion of Eclipse window (in the console section).



5. Now that you can successfully run a program in Eclipse, let us get back to our experiment. Replace the "System.out.println" line with the follow piece of code and rerun your code. Alternatively, copy CoinFlipExperiment.java available in part3 folder and place it in your workspace folder. That is: `workspace/Lab01/src/lab01)`

```
double flip = Math.random();
if (flip < 0.505) {
  System.out.println ("Heads");
}
else {
  System.out.println ("Tails");
}
```

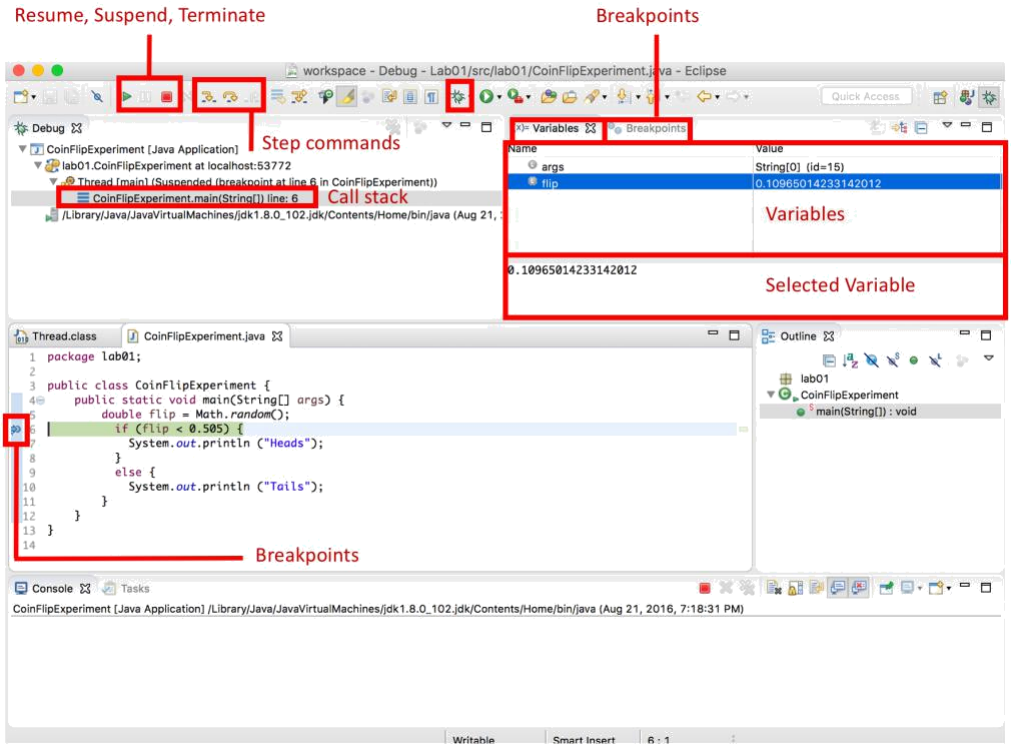
6. Run your program multiple times and see the output.
7. Now, let us understand this piece of code. In the code above, the Math.random() method generates a random number between [0...1). If the random number is less than 50.5% (ie. 0.505), we treat it as “heads”, otherwise it’s “tails”.
8. Run your program a few times to make sure the coin flipping works. (Note, it is not unlikely for it to be the same five times in a row. It would be very unlikely to be the same 10 times in a row. Make sure you run it until you see both results.)

Part 3 – Debugging

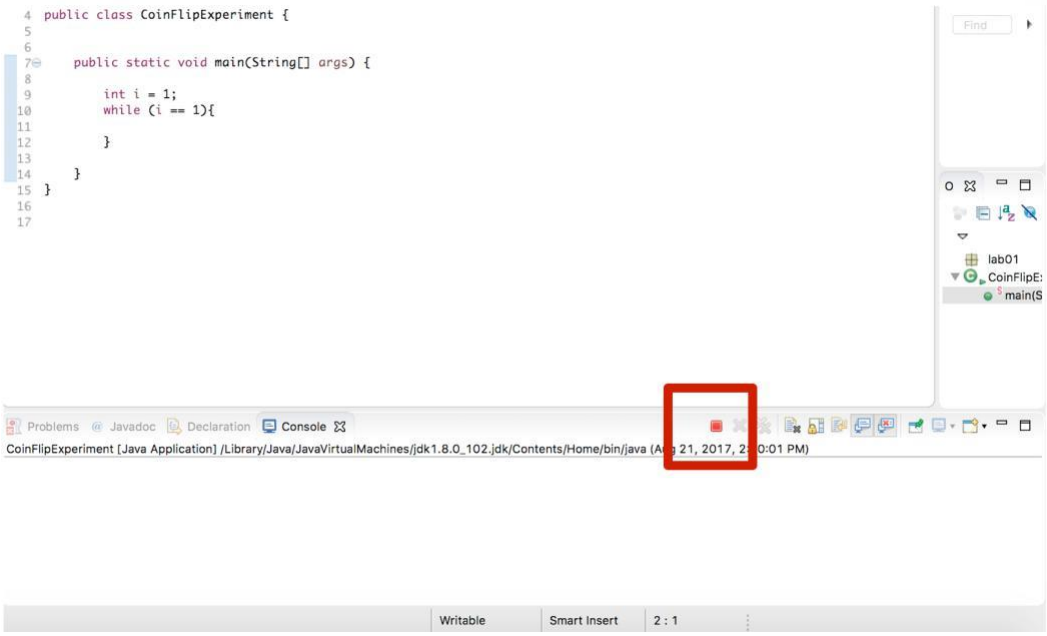
In this part, we are going to introduce you to Eclipe’s debugger tool, and see how to use it to enhance your understanding of what your code is doing.

One of the most useful tools in Eclipse for debugging is a “breakpoint” so that we can stop the program execution on a specific line.

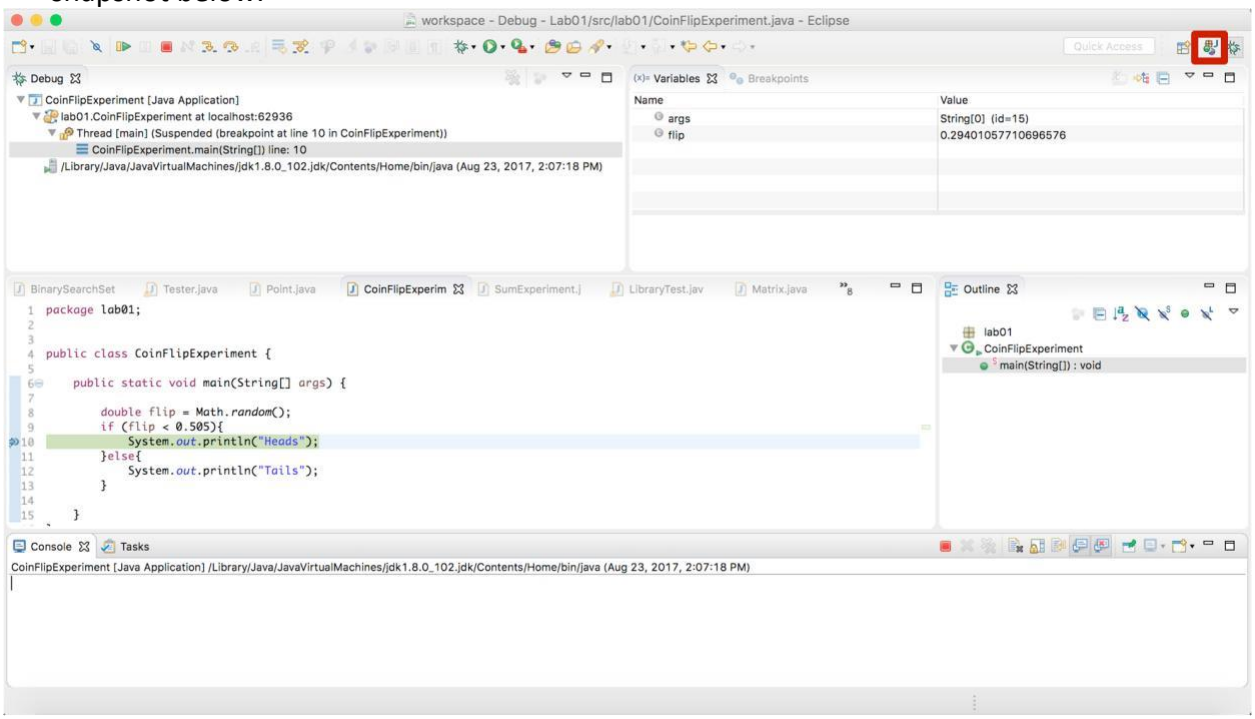
1. To create a breakpoint, double click on the line where you have the “if” statement. You will see a small circle appear on that line.
2. Now, if you run your program as before you won’t see any difference, to stop the execution on this line you need to run the program in “debug mode”.
3. Instead of running the program by clicking the “run” button, click on the button on its left (the little bug). A window opens up, asking you whether you want to switch to “debug perspective”. Click “yes”, and see what happens.
4. Now, you will see a window that looks like the following figure. Please spend a couple of minutes understanding various tabs you are seeing in this figure before you go back to Eclipse.



5. Now you can go back to Eclipse and try different functionalities of the debug view.
6. Notice the red square on your console. When your program is done running this should turn into a grey square, if it stays red that means you have an infinite loop. You will be penalized at least 20% if you have any infinite loop in your assignment. Watch for it! The following shows you a snapshot of a simple infinite loop (you can also try it).



7. To get back to the original view of the eclipse window, you need to first “terminate” the program by clicking the little cross sign close to the red square (on the console window at the bottom) and then click on the little button on the top right - marked in the snapshot below.



Part 4 – Make Your Coin Experiment more interesting

Modify the code for coin flip experiment so that we flip the code for 100 times (using a loop), and keep track of how much money you win or lose. Here is an outlie how to do it to save you time:

```
// Declare an int variable to keep track of winnings.
// Set winnings to 0.
// Loop 100 times.
  // Flip the coin – reuse the code you wrote in the lab
  // If heads, add $1 to winnings.
  // If tails, subtract $1 from winnings.
// Print out winnings.
```

Start by modifying the code in part 3 of lab 1 in blackboard.

Run your code a few times and make sure it is working. Remember that you will sometimes win and sometime lose…

The next step is, instead of doing everything in the "main", write a function that keeps track of the amount. Here is how the outline of your method should look like:

```
/** Returns the amount of money you'd win or lose
  * by flipping an unbalanced coin 100 times.
*
  * @return the amount of money won/lost
*/
  static public int coinFlipExperiment ()
  {
        // finish this function: it should be as easy as moving around some code
  }
```

Next, rearrange your program so that the coin flipping is inside the method, the method returns the winnings, and "main" contains only this code:

```
  int amount = coinFlipExperiment ();
  System.out.println ("Win/loss amount: " + amount);
```

In other words, the code you are going to submit should look like this – obviously you will finish the function.

```
public class CoinFlipExperiment {

    static public int coinFlipExperiment (){
        // You will finish write this function
    }


    public static void main(String[] args) {
        int amount = coinFlipExperiment ();
          System.out.println ("Win/loss amount: " + amount);
    }
}
```

Run your program a few times to make sure it is working and again remember you will see different values every time and that is okay!

After you are done you can submit your code, upload it to your university box account (following the assignment submission guideline in blackboard).

Part 6 – Upload Your Code in Your University Box Account

Please consult the box tutorial uploaded in blackboard. The lab instructor will help you if you have any questions.

Part 7 – Start your first assignment

You are not required to finish this part during the lab but I highly recommend you get started on it during the lab.

For your first assignment, imagine that you are given an array of integer values. The values in the array are sorted in an ascending order and you want to figure out whether there exists any pair of numbers (i.e., any two elements) in this sorted array that will add up to 20. You need to write a simple function that perform this task. For simplicity, you can assume if there exists such as pair, it is unique. Please follow the following steps exactly to get started:

1. In your lab01 package, create a class (or Java file) as you learned and call it: **SumExperiment.java** (nothing else!).

2. Log into Blackboard and copy the provided **SumExperiment.java** file into your src folder (or copy the content in the file you just created). You should now see a **main** function

and another function called **checkSum**.

3. As you can see the **checkSum** function, receives an array of integers as its input and returns an integer value. The return value is the **index** for the smaller value in the pair. If such a pair does not exist in the array, the function is supposed to return -1. For instance, if we give the following array to **checkSum** function: {5, 7, 8, 9, 10, 15, 16}, **checkSum** should return 0. Why?

4. IMPORTANT NOTE: DO NOT CHANGE THE SIGNATURE OF THE FUNCTION OR THE JAVA FILE. IF YOU CHANGE THE SIGNATURE OF THE FUNCTION (ITS NAME, INPUT OR OUTPUT OR THE FILE NAME) YOU WILL LOSE POINTS (rational: we use automated scripts for grading so if you change things, the grading will become tedious for the TA who is a student just like yourself!)

5. Now start thinking about how would you accomplish this task. Take a moment to come up with a solution on your own before you go to the next step!

6. The naïve way to accomplish this task is to check all the pairs in the array. However, remember that the array is sorted, can we somehow deploy this property do this task more quickly? Again, try to think about it before you look at the next step. Hint: think about how you can use two pointers (or iterators) to accomplish this task more efficiently than looking at all pairs.

7. Okay, hopefully you already got the idea. If not, I strongly encourage you to stop reading and think about it – I will NOT provide the idea every time ;)

8. Here is how you can do it more efficiently: Imagine you have two pointers (they can simply be two integer values that keep the index of the array), one pointing to the beginning of the array and advances forward and the other one points to the last element of the array and advances backward. All you need to do is to sum the values these two pointers point to and see if they are equal to 20, if so, great! you can prepare the output of the function.

9. If not, you inspect the sum, if the sum is greater than 20, you will decrement the second pointer and if the sum is less than 20, you increment the first pointer.

10. Before start coding, try the idea on a piece of paper with some example (you can use the ones provided in the main function) and convince yourself that this would work.

11. Now, start writing the body of the function. Again, do not change the signature!

12. After you are finished writing the function, you can use the main function provided to test your code. You see that we have four arrays in there. Try to workout each case in your mind (or with pen and paper) and understand what those "if" statements are going to do.

13. If you see any "TEST FAIL" when you run your program, something is wrong with your function. Go back and debug your function. If your function is working properly, you show only see "Done!!!!". If you see anything else, something is wrong!

Make sure to upload your code to Box before your leave the lab if you want to work on your personal computer at home. If you need to install Eclipse consult the tutorial uploaded in blackboard.

For all your assignments, please start early and seek help early (either from the instructor or the TAs).