North South University
Department of Electrical and Computer Engineering

CSE 215 (Both Theory and Lab): Assignment 02 (Problem Set)
Course Instructor: Dr. Mohammad Rezwanul Huq

Problem Solving on Loops, Methods and Arrays

Important Instructions:

a. Class name of the solution of the problems should be like Problem1, Problem2 and so on.

b. Create a zip/rar file which contains the solution **(.java file only)** of the problems only. As an example, only include Problem1.java, Problem2.java and so on.

c. The file name must follow the format:
<section number>_<assignment number>_<student number>.
Example: 13_01_1710000042. Files with incorrect naming will not be accepted.

d. Use the link below to upload the zip/rar file.
http://bit.ly/cse215codesubmit

Submission Deadline:

06 July 2019 11:59PM.

Problem Set

1. (*Count positive and negative numbers and compute the average of numbers*) Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input **0**. Display the average as a floating-point number. Here is a sample run:

```
Enter an integer, the input ends if it is 0: 1 2 -1 3 0 Enter
The number of positives is 3
The number of negatives is 1
The total is 5.0
The average is 1.25
```

```
Enter an integer, the input ends if it is 0: 0 Enter
No numbers are entered except 0
```

2. (*Display pyramid*) Write a program that prompts the user to enter an integer from **1** to **15** and displays a pyramid, as shown in the following sample run:

```
Enter the number of lines: 7 Enter
                    1
                  2 1 2
                3 2 1 2 3
              4 3 2 1 2 3 4
            5 4 3 2 1 2 3 4 5
          6 5 4 3 2 1 2 3 4 5 6
        7 6 5 4 3 2 1 2 3 4 5 6 7
```

3. (*Sum a series*) Write a program to sum the following series:

$$\frac{1}{3}+\frac{3}{5}+\frac{5}{7}+\frac{7}{9}+\frac{9}{11}+\frac{11}{13}+\cdots+\frac{95}{97}+\frac{97}{99}$$

4. (*Decimal to binary*) Write a program that prompts the user to enter a decimal integer and displays its corresponding binary value. Don't use Java's **Integer.toBinaryString(int)** in this program.

5. (*Palindrome integer*) Write the methods with the following headers:

   // Return the reversal of an integer, i.e., reverse(456) returns 654
   public static int reverse(**int** number)
   // Return true if number is a palindrome
   public static boolean isPalindrome(**int** number)
   Use the **reverse** method to implement **isPalindrome**. A number is a palindrome if its reversal is the same as itself. Write a test program that prompts the user to enter an integer and reports whether the integer is a palindrome.

   (Convert the given integer into a String by writing String n = number + ""; Then use a loop to get each character using n.charAt(i) function and build the reverse string. Later convert the reverse string into an integer using Integer.parseInt(reverse) function.)

6. (*Estimate* π) π can be computed using the following series:

$$m(i)=4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots+\frac{(-1)^{i+1}}{2i-1}\right)$$

   Write a method that returns **m(i)** for a given **i** and write a test program that displays the following table:

i	m(i)
1	4.0000
101	3.1515
201	3.1466
301	3.1449
401	3.1441
501	3.1436
601	3.1433
701	3.1430
801	3.1428
901	3.1427

7. (*Palindromic prime*) A *palindromic prime* is a prime number and also palindromic. Use a function to determine prime numbers and another function to determine whether it is palindromic or not.

For example, 131 is a prime and also a palindromic prime, as are 313 and 757. Write a program that displays the first 100 palindromic prime numbers. Display 10 numbers per line, separated by exactly one space, as follows:

2 3 5 7 11 101 131 151 181 191
13 353 373 383 727 757 787 797 919 929

8. (*Count occurrence of numbers*) Write a program that reads the integers between 1 and 100 and counts the occurrences of each. Assume the input ends with **0**. Here is a sample run of the program:

```
Enter the integers between 1 and 100: 2 5 6 5 4 3 23 43 2 0 Enter
2 occurs 2 times
3 occurs 1 time
4 occurs 1 time
5 occurs 2 times
6 occurs 1 time
23 occurs 1 time
43 occurs 1 time
```

Note that if a number occurs more than one time, the plural word "times" is used in the output.

9. (*Print distinct numbers*) Write a program that reads in ten numbers and displays the number of distinct numbers and the distinct numbers separated by exactly one space (i.e., if a number appears multiple times, it is displayed only once). (*Hint*: Read a number and store it to an array if it is new. If the number is already in the array, ignore it.) After the input, the array contains the distinct numbers. Here is the sample run of the program:

```
Enter ten numbers: 1 2 3 2 1 6 3 4 5 2 Enter
The number of distinct number is 6
The distinct numbers are: 1 2 3 6 4 5
```

10. Write a program to compute the standard deviation using the following formula.

$$mean = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n} \quad deviation = \sqrt{\frac{\sum_{i=1}^{n} (x_i - mean)^2}{n - 1}}$$

To compute the standard deviation with this formula, you have to store the individual numbers using an array, so that they can be used after the mean is obtained.

Your program should contain the following methods:

```
/** Compute the deviation of double values */
public static double deviation(double[] x)

/** Compute the mean of an array of double values */
public static double mean(double[] x)
```

Write a program that prompts the user to enter ten numbers and displays the mean and standard deviation, as shown in the following sample run:

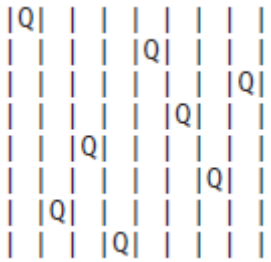
```
Enter ten numbers: 1.9 2.5 3.7 2 1 6 3 4 5 2 Enter
The mean is 3.11
The standard deviation is 1.55738
```

11. (*Sort students*) Write a program that prompts the user to enter the number of students, the students' names, and their scores, and prints student names in decreasing order of their scores.

12. (*Execution time*) Write a program that randomly generates an array of 100,000 integers within 0 to 99999. Prompt the user to enter the value he/she wants to search for. Estimate the execution time of invoking the **linearSearch** method as given in the textbook (in Listing 7.6). Sort the array using **selection sort** algorithm and estimate the execution time of invoking the **binarySearch** method in Listing 7.7. You can use the following code template to obtain the execution time:

```
long startTime = System.currentTimeMillis();
//perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;
```

13. (*Game: Eight Queens*) The classic Eight Queens puzzle is to place eight queens on a chessboard such that no two queens can attack each other (i.e., no two queens are on the same row, same column, or same diagonal). There are many possible solutions. Write a program that displays one such solution. A sample output is shown below:



14. (*Sort characters in a string*) Write a method that returns a sorted string using the following header:

```
public static String sort(String s)
```

For example, **sort("acb")** returns **abc**.

Write a test program that prompts the user to enter a string and displays the sorted string.

15. (*Strictly identical arrays*) The arrays **list1** and **list2** are *strictly identical* if their corresponding elements are equal. Write a method that returns **true** if **list1** and **list2** are strictly identical, using the following header:

```
public static boolean equals(int[] list1, int[] list2)
```

Write a test program that prompts the user to enter two lists of integers and displays whether the two are strictly identical. Here are the sample runs. Note that the first number in the input indicates the number of the elements in the list. This
number is not part of the list.