**North South University**
**Department of Electrical and Computer Engineering**

**CSE 215: Assignment 02 (Problem Set)**
**Course Instructor: Dr. Mohammad Rezwanul Huq**

# Problem Solving on Loops, Methods, Arrays

Submission Instructions:

- Write the solution of each problem named as A02_P01.java and so on.
- Zip all your .java files. Remember, only the java files. Name the zip file according to your student id such as 1911234042.zip.
- Upload the zip file in the following link. https://goo.gl/m99ftH (Select Assignment 02).
- Solution must be uploaded by **Sunday 11:59:59 PM, 17 March 2019.**
- **Failure to submit the weekly assignment will result into deducting marks from both theory and lab performance.**

## Problem Set

1. (*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
```

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

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

3. (*Display calendars*) Write a program that prompts the user to enter the year and first day of the year and displays the calendar table for the year on the console. For example, if the user entered the year **2013**, and **2** for Tuesday, January 1, 2013, your program should display the calendar for each month in the year, as follows:

**January 2013**

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | | |

December 2013

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 | | | | |

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.)

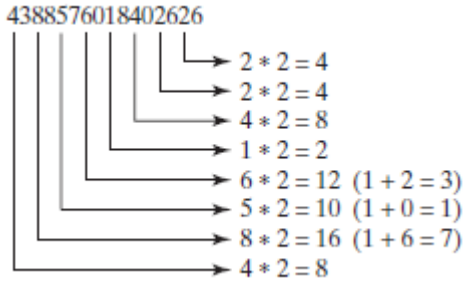
6. (*Palindromic prime*) A *palindromic prime* is a prime number and also palindromic. Use a function to determine prime numbers and another function to determine whether it is palindromic or not.
For example, 131 is a prime and also a palindromic prime, as are 313 and 757. Write a program that displays 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
```

7. (*Financial: credit card number validation*) Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:
   - 4 for Visa cards
   - 5 for Master cards
   - 37 for American Express cards
   - 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check*, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.




2. Now add all single-digit numbers from Step 1.
   4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37
3. Add all digits in the odd places from right to left in the card number.
   6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38
4. Sum the results from Step 2 and Step 3.
   37 + 38 = 75
5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a **long** integer. Display whether the number is valid or invalid. Design your program to use the following methods:

```
/** Return true if the card number is valid */
public static boolean isValid(long number)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)

/** Return this number if it is a single digit, otherwise, return the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number)

/** Return true if the digit d is a prefix for number */
public static boolean prefixMatched(long number, int d)

/** Return the number of digits in d */
public static int getSize(long d)

/** Return the first k number of digits from number. If the number of digits in number is less than k,
return number. */
public static long getPrefix(long number, int k)
```

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.You may want to use charAt() and Integer.parseInt() function in your code)

```
Enter a credit card number as a long integer:
  4388576018410707 Enter
4388576018410707 is valid
```

```
Enter a credit card number as a long integer:
  4388576018402626 Enter
4388576018402626 is invalid
```

8. (*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.

9. (*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;
```

10. (*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.