Exercise 1

Write a method `isPalindromePhrase` that accepts a string as argument and returns `true` or `false` indicating if the string is a palindrome or not. In this version of the method, whitespaces, commas, and apostrophes are ignored. For example, "Lonely Tylenol" and "Madam, I'm Adam" are both palindromes. The signature of the method should be:

```
public static boolean isPalindromePhrase(String str)
```

You are not allowed to generate a new string in your implementation of this method, and your iteration must be expressed with a **`while`** loop. Use the method `isPalindromePhrase` to write a program **`Palindromes.java`** that asks the user to enter an integer *N* using the scanner object followed by *N* strings and prints the strings that are palindromes according to this extended definition.

Exercise 2

Write a program **`Discount.java`** that reads in a list of prices from command line arguments and writes on the standard output a ***formatted*** table consisting of three columns as shown below. The first column consists of the prices as read from the command line (in US dollars). The second column prints a discounted price that represents a 1/3 discount on the price. The third column prints the equivalent value in Lebanese Liras rounded to the nearest LL 50 (assume a conversion rate $1 = LL 1508). Your output should have 2 digits after the decimal in the first 2 columns and no decimal in the third. It should also print column headers and makes sure the columns are lined up properly. An example invocation of the program might produce:

```
> java Discount 2 80.0 140.0
Price (in $USD)     Discounted    Equivalent Price (in LL)
$ 80.00             $53.33        LL  80450
$140.00             $93.33        LL 140750
```

Exercise 3

Say that you owe Bank Audi *X* dollars. The bank charges you an interest of *P* percent per month on the unpaid balance. You have decided to stop using the card and to pay off the debt by making a monthly payment of *N* dollars a month. Write a program **`BankAudi.java`** that writes out the balance and total payments so far for every succeeding month until the balance is zero or less.

For each month, calculate the interest due on the unpaid balance; then calculate the new balance by adding the interest and subtracting the payment. For Example, for a bill of $1000.0, interest rate of 1.5% and monthly payment of $100.0, the program's output will be as follows (**you must properly format your output as shown**):

```
> java CreditCard 1000.0
Enter the monthly payment (in $): 100.0
Enter the interest rate (as percentage): 1.5
Month: 1     balance: 915.00     total payments: 100.0
Month: 2     balance: 828.73     total payments: 200.0
Month: 3     balance: 741.16     total payments: 300.0
Month: 4     balance: 652.27     total payments: 400.0
Month: 5     balance: 562.06     total payments: 500.0
Month: 6     balance: 470.49     total payments: 600.0
Month: 7     balance: 377.55     total payments: 700.0
Month: 8     balance: 283.21     total payments: 800.0
Month: 9     balance: 187.46     total payments: 900.0
Month: 10    balance: 90.27      total payments: 1000.0
Month: 11    balance: -8.38      total payments: 1100.0
```

Exercise 4

Write an interactive program **`GuessTheNumber.java`** that prompts the user to think of an integer between one and a million and then asks the user to truthfully respond with true or false to statements of the form "*My number is less than or equal to x?"* for various values of *x*, until the program can deduce the chosen integer. Your program should use a strategy known as binary search for guessing. Binary search chooses the middle of the search interval as the value of x at every iteration.

Exercise 5

Write a program **`MaximumLocalAlpha.java`** that prompts the user to enter a sequence of positive numbers, which your program should operate on, in order to find the maximum *local Alpha* in the sequence. A value is a *local Alpha* if it is larger than both the preceding and succeeding values in the sequence. Note: Neither the first nor the last values of the sequence can be local *Alpha's*.

For example, for the sequence `4 2 1 6 8 13 11 6 12 5`, there are two local *Alpha's*: 13 and 12; because they are both larger than their neighbors (i.e., 8 and 11; and 6 and 5, respectively). Your program should only output the maximum local *Alpha*; thus in this case 13. The following is a sample run of the program (the user input via Scanner is underlined):

```
> java MaximumLocalAlpha
Please enter a sequence of positive integers: 4 2 1 6 8 13 11 6 12 5
Maximum Local Alpha = 13
```

Submission Instructions

- Your submission must consist of a single zip folder that contains .java files only (**`Palindromes.java`**, **`Discount.java`**, **`BankAudi.java`**, **`GuessTheNumber.java`**, and **`MaximumLocalAlpha.java`**). No additional files should exist in the .zip folder.
- Give meaningful names to your methods and variables in your code.
- Include a comment at the beginning of your program with basic information about yourself and a description of the program. Include also a comment at the start of each method.
- The name of the zip file must adhere to the following naming convention s*#*_A$_*netid*, where # stands for your section number (between 1 and 12), $ stands for the assignment number, and *netid* stands for your AUBnet user name. For example, if your AUBnetid is abc65 and you are in section 4, you should submit Assignment 8 using following file: s4_A8_abc65.zip. The zip files will be processed automatically so please make sure you use this naming convention.
- **Failing to follow these guidelines will result in deducting marks form your grade.**