Lab Questions

Note: All questions except the last two need to solved using recursion.

1. Write a *recursive* function that takes in one argument n and computes $n!$, the factorial function. Recall that

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

2. Write a *recursive* function '`count7()`' that given a non-negative integer `n`, returns the count of the occurrences of `7` as a digit, so for example `count7(717)` yields `2`. Following are some more examples

   - `count7(7170123)` should return 2
   - `count7(7)` should return 1
   - `count7(123)` should returns 0

   Hint: Note that mod (%) by 10 yields the rightmost digit (126%10 is 6), while divide (/) by 10 removes the rightmost digit (126/10 is 12).

3. Write a recursive function `isPalindrome()` that returns **`true`** if and only if the input `string` is palindrome (a palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as **"Madam"** or **"Racecar"** or the number **"10201"**.)

   Hint: Ensure that the first and the last letters are the same, and then recursively check the rest of the string. Use the `substr()` function to get the substring of a given string (examples at the end).

4. Write a recursive method to find maximum elements in a given array. Implement the following recursive function.

```
public static int findMax(int[] a) {
  return findMax(a, a.length);
}

// This helper function finds the maximum of first n elements in the array
public static int findMax(int[] a, int n) {
    if(n==____)
        return ________; // base case

    int m = ___________; // recursively find the maximum of n-1 elements

    return _____________; // return maximum of m and the last element
}
```

5. Write a recursive method `sumTo()` that accepts an integer parameter `n` and returns the sum of the first `n` reciprocals. In other words: `sumTo(n)` returns: $1 + 1/2 + 1/3 + 1/4 + \cdots + 1/n$. Assume that your method is always passed a value greater than equal to `0`.

6. Write a recursive method `writeChars()` that accepts an integer parameter `n` and that prints out `n` characters as follows. The middle character of the output should always be an asterisk (`"*"`). If you are asked to write out an even number of characters, then there will be two asterisks in the middle (`"**"`). Before the asterisk(s) you should write out less-than characters (`"<"`). After the asterisk(s) you should write out greater-than characters (`">"`). See the following for some example calls along with the output.

Call	Output
`writeChars(1)`	`*`
`writeChars(2)`	`**`
`writeChars(3)`	`<*>`
`writeChars(4)`	`<**>`
`writeChars(5)`	`<<*>>`
`writeChars(6)`	`<<**>>`
`writeChars(7)`	`<<<*>>>`
`writeChars(8)`	`<<<**>>>`

   Assume that your method is always passed a value greater than equal to 1.

7. Write a function `isValid()` to check whether a given string is a valid password, i.e., it obeys the following rules:

   - A password must have at least ten characters.
   - A password consists of only letters and digits.
   - A password must contain at least two digits.
   - A password must contain at least one letter.

   A reference of useful String operations and other functions is provided at the end.

8. Two strings are called anagrams if they contain same set of characters but in different order. Some interesting examples are:

   - "Debit card" = "Bad credit",
   - "Graduation" = "Out in a drag!",
   - "Election Results" = "Lies, Let's Recount!",
   - "Software" = "Swear Oft".

   Write a function `isAnagram()` which checks whether two given strings are anagrams of each other.

Strings and related functions

The following functions can be used to check if a character is a digit, letter, etc. You need to include the header file `<cctype>` to use these functions.

function	description
`isalnum(c)`	checks if `c` is alphanumeric (either a digit or a letter)
`isalpha(c)`	checks if `c` is a letter
`isdigit(c)`	checks if `c` is a digit
`islower(c)`	checks if `c` is a lowercase letter
`isupper(c)`	checks if `c` is an uppercase letter

To compute a *substring* of a given `string str`, the `substr()` function can be used. The first argument of the function is the starting index of the substring, and the second argument is the length of the substring. Here are few examples (using `str = "Hello World"`):

- `str.substr(1)` returns a substring of `str` starting from index `1` till the end of the string. `"ello World`
- `str.substr(1, 3)` returns a substring of `str` starting from index `1` and of length `3`. `"ell`
- `str.substr(1, str.length()-2)` returns a substring of `str` starting from index `1` and of length `str.length()-2`. `"ello Worl"`