CSC111 Lab 6 – Recursion Fall 2017

Found on the Internet:

> This is the song that doesn't end,
> It just goes on and on my friend,
> Some people started singing it not knowing what it was,
> And they'll continue singing it forever just because,
> This is the song doesn't end ...

Also Found on the Internet:

How to "find your way home"

1. If you are at home, stop moving.
2. Take one step toward home.
3. "find your way home".

In this lab we will explore recursion. Rather than one complex project, we will begin gently with a progressive series of examples which are solved using recursion.

First Recursive Problem

The <u>factorial</u> function should be familiar to you from your study of mathematics. An exclamation mark is used to denote factorial. For example:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

In general:

$$n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1$$

By definition, $0! = 1$. We can think of the factorial recursively by observing:

$$n! = n \times (n-1)!$$

Your task: Write a Java program that accepts a non-negative integer on the command line and prints the factorial. Your program must include a **recursive** static method as follows:

```
static int factorial( int n )
{
      // Your implementation here.
}
```

For example, a program run and the output are shown below:

```
gottlieb% java main 5
120
```

Second Recursive Problem

Given two positive integers A and B, the greatest common divisor (acronym: **gcd**) is a number d such that:

1. d divides both A and B
2. Every divisor of both A and B also divides d.

The **gcd** of two numbers can be computed recursively by the following two rules:

$$\begin{aligned} \gcd(A,0) &= A \\ \gcd(A,B) &= \gcd(B, A \text{ modulo } B) \end{aligned}$$

Your task: Write a Java program that accepts two numbers on the command line and outputs the greatest common divisor. Your program must include a **recursive** static method as follows:

```
static int gcd( int A, int B )
{
      // Your implementation here.
}
```

For example, a program run and the output are shown below:

```
gottlieb% java main 42 1155
21
```

Third Recursive Problem The number of distinct binary trees with $n+1$ nodes is given by the $n^{\underline{\text{th}}}$ Catalan number, C_n. The Catalan numbers are defined by the equations:

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i\, C_{n-i} \quad \text{for } n \geq 0; \tag{1}$$

The first few Catalan numbers for $n = 0, 1, 2, 3, ..., 10$ are

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796$$

Your task: Write a Java program that accepts one number n on the command line and outputs the $n^{\underline{\text{th}}}$ Catalan number, C_n.

Your program must include a **recursive** static method as follows:

```
static int catalan( int n )
{
      // Your implementation here.
}
```

For example, a program run and the output are shown below:

```
gottlieb% java main 5
42
```

Obviously, “The answer is 42”.

From Wikipedia:

> In “The Hitchhiker’s Guide to the Galaxy”, a group of hyper-intelligent pan-dimensional beings demand to learn the Answer to the Ultimate Question of Life, The Universe, and Everything from the supercomputer, “Deep Thought”, specially built for this purpose. It takes Deep Thought 7.5 million years to compute and check the answer, which turns out to be 42. Deep Thought points out that the answer seems meaningless because the beings who instructed it never actually knew what the Question was.

Fourth Recursive Problem

Searching an ordered list can be done very efficiently by a method known as **binary search**. Suppose we have the following array with 13 elements in it:

2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**	**31**	**37**	**41**
0	1	2	3	4	5	6	7	8	9	10	11	12

Suppose also we are looking for the number 29. We could search one position at a time, but it is much faster if we look in the middle of the list.

$$\left\lfloor \frac{13}{2} \right\rfloor = 6$$

In position 6 we find the number 17. Our target number, 29 is larger than 17, so we may discard the left half of the array. The search for 29 continues in the right half of the array.

19	**23**	**29**	**31**	**37**	**41**
7	8	9	10	11	12

Your task: Implement binary search recursively. Write a static method `search` with the following header:

```
static int search( int [] A, int target, int left, int right )
{
      // Your implementation here.
}
```

Your search method should return the array index where the search target is found. For example, when searching for 29, your method should return the number 9. If the search target is not in the array, then return -1.

You can download a main program for this part from **menehune**.

Search Logic:

1. Base case: `if ( right < left )` .
   If, at some point in the search, the right boundary of the search space becomes less than the left boundary, there are no remaining places to look. The search fails. Return -1.

2. General case:

   (a) Compute `mid = (left + right) / 2 ;`

   (b) if `target == A[mid]`, then the search is successful. Return `mid`.

   (c) if `target < A[mid]`, then the target must be to the left of the `mid` position. Recursively search in the range [`left`, `mid-1`]. Return whatever value the recurive call returns.

   (d) if `target > A[mid]`, then the target must be to the right of the `mid` position. Recursively search in the range [`mid+1`, `right`] Return whatever value the recurive call returns.

Turn In;

Save all your work in a directory named "Lab6". Change to your home directory (the parent directory of "Lab6"), and create a file named "lab6.tar" using the command:

```
tar cf lab6.tar Lab6
```

Use `sftp` to upload the file "lab6.tar" to your account on telesto.