2018

Program III

EJERCICIOS DE RECURSIVIDAD

Tabla de contenido

factorial

Given n of 1 or more, return the factorial of n, which is n * (n-1) * (n-2) ... Compute the result recursively (without loops).

factorial(1) → 1
factorial(2) → 2
factorial(3) → 6

```
public int factorial(int n) {

}
```

fibonacci

The fibonacci sequence is a famous bit of mathematics, and it happens to have a recursive definition. The first two values in the sequence are 0 and 1 (essentially 2 base cases). Each subsequent value is the sum of the previous two values, so the whole sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21 and so on. Define a recursive fibonacci(n) method that returns the nth fibonacci number, with n=0 representing the start of the sequence.

fibonacci(0) → 0
fibonacci(1) → 1
fibonacci(2) → 1

```
public int fibonacci(int n) {

}
```

count7

Given a non-negative int n, return the count of the occurrences of 7 as a digit, so for example 717 yields 2. (no loops). 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).

count7(717) → 2
count7(7) → 1
count7(123) → 0

```
public int count7(int n) {

}
```

countX

Given a string, compute recursively (no loops) the number of lowercase 'x' chars in the string.

countX("xxhixx") → 4
countX("xhixhix") → 3
countX("hi") → 0

```
public int countX(String str) {

}
```

strCount

Given a string and a non-empty substring **sub**, compute recursively the number of times that sub appears in the string, without the sub strings overlapping.

strCount("catcowcat", "cat") → 2
strCount("catcowcat", "cow") → 1
strCount("catcowcat", "dog") → 0

```
public int strCount(String str, String sub) {

}
```

noX

Given a string, compute recursively a new string where all the 'x' chars have been removed.

noX("xaxb") → "ab"
noX("abc") → "abc"
noX("xx") → ""

```
public String noX(String str) {

}
```

powerN

Given **base** and **n** that are both 1 or more, compute recursively (no loops) the value of base to the n power, so powerN(3, 2) is 9 (3 squared).

powerN(3, 1) → 3
powerN(3, 2) → 9
powerN(3, 3) → 27

```
public int powerN(int base, int n) {

}
```

strCopies

Given a string and a non-empty substring **sub**, compute recursively if at least n copies of sub appear in the string somewhere, possibly with overlapping. N will be non-negative.

strCopies("catcowcat", "cat", 2) → true
strCopies("catcowcat", "cow", 2) → false
strCopies("catcowcat", "cow", 1) → true

```
public boolean strCopies(String str, String sub, int n) {

}
```

changeXY

Given a string, compute recursively (no loops) a new string where all the lowercase 'x' chars have been changed to 'y' chars.

changeXY("codex") → "codey"
changeXY("xxhixx") → "yyhiyy"
changeXY("xhixhix") → "yhiyhiy"

```
public String changeXY(String str) {
}
```

allStar

Given a string, compute recursively a new string where all the adjacent chars are now separated by a "*".

allStar("hello") → "h*e*l*l*o"
allStar("abc") → "a*b*c"
allStar("ab") → "a*b"

```
public String allStar(String str) {

}
```

countPairs

We'll say that a "pair" in a string is two instances of a char separated by a char. So "AxA" the A's make a pair. Pair's can overlap, so "AxAxA" contains 3 pairs -- 2 for A and 1 for x. Recursively compute the number of pairs in the given string.

countPairs("axa") → 1
countPairs("axax") → 2
countPairs("axbx") → 1

```
public int countPairs(String str) {

}
```