CSC-363: Problem Set I: Language Characteristics I

1. For the `Point` class below, define an `equals` method (that takes an `Object` as parameter and returns true or false) whether the given object is equal in its coordinates with `this` object; assume a Java implementation.

```
class Point {
   int dimensions;
   float coordinates[];
   Point () {
      dimensions = 2;
      coordinates = new float[2];
   }
   Point (int d) {
      dimensions = d;
      coordinates = new float[d];
   }
}
```

2. Rewrite the equals method from question **1** using the `as` operator where

`object as Class`

is interpreted according to:

`object instanceof Class ? (Class) object : null`.

C# defines the `as` operator in this manner.

3. In C / C++, the `static` keyword can be applied to a local variable in a function. `static`, local variables retain the value of the variable between function calls, but remains visible only to the function or block in which it is defined.

The limited visibility of static local variables facilitate a form of encapsulation (data hiding) in a non-object-oriented language while also maintaining values across calls. This allows us to safely use a technique called *memoization*: after we successfully compute a value for a given input, we save the result (to avoid recomputing it in the future).

Write function `fibonacci` in C (not C++) that computes the nth Fibonacci number according to the recurrence relation:

`fibonacci(0) == 0`,

`fibonacci(1) == 1`, and

`fibonacci(n) == fibonacci(n-1) + fibonacci(n-2)`.

4. Consider a list class in Java that stores values with a primitive array (somewhat like `ArrayList`); our class will be called `SliceableList<T>`. Consider a modification to the Java language allowing operator overloading by defining a method using a keyword `operator` or `binary-operator` and syntax like the following:

```
<visibility> <return-type> operator<the-operator>(<arguments>).
```

Our goal in this problem is to define a slicing operator for our class:

```
public SliceableList<T> binary-operator :: (int beginIndex, int endIndex)
```

In this case `beginIndex` is inclusive and `endIndex` is exclusive to the slice. Do not use any library functionality (i.e., stick to the fundamentals by using loops and primitive arrays). Assume only positive indices. If the indices are not ordered properly, return an empty list. Do not assume any method implementations; if need be, provide support method implementations.