# CS 538

## Project #4

## Programming in Java, Pizza and Python

Due Friday, May 10, 2002

Not accepted after Monday, May 13, 2002

This assignment is not required; you may do any or all of the following three programs for **extra credit.**

1. You are to redo question 4 of Assignment #2 (lazy lists and the "Sieve of Erastosthenes") in Java. Since Java is not polymorphic, you will implement lazy lists of type `Object`. Since Java does not support first-class functions, you'll need to use the following "trick" to encapsulate a function within a class definition. Assume we wish to pass a function of type `() -> LazyList`. We'll first create an abstract class that contains only one member function, `f`:

```
abstract class LazyListFct {
     abstract LazyList f();
}
```

All subclasses of `LazyListFct` will simply redefine `f` to be a particular `() -> LazyList` function, e.g.

```
class SeqTail extends LazyListFct{
      LazyList f() { return seq(s,l); }
}
```

Whenever a particular function must be passed as a value, an instance of a subclass of `LazyListFct` is used. The class you need to implement, `LazyList`, is structured as follows:

```
class LazyList {
   boolean NullList; // Is this list null?
   Object head;      // Head of this list, if NullList is false
   LazyListFct tail; // Function to compute tail of this list,
                     // if NullList is false
   LazyList() { NullList = true;} // Creates a null LazyList
   LazyList(Object h, LazyListFct t) {
      NullList = false; // Creates a non-null LazyList
      head = h;
      tail = t;
   }
   static LazyList seq(int start, int finish){
         // Same def as in Question #4 of project 2
```

```
        }

        static LazyList infseq(int start){
              // Same def as in Question #4 of project 2
        }
        static LazyList boolseq(boolean start){
              // Creates an infinite LazyList containing the values
              // start, !start, start, !start, ...
        }
        static LazyList constList(int val){
              // Create an infinite LazyList containing val in
              //  every position
        }
        static LazyList filter(LazyList control, LazyList data){
              // Same def as in Question #4 of project 2
        }
        static LazyList primes() {
              // Same def as in Question #4 of project 2
        }
        Object Nth(int n) {
              // Same def as in Question #4 of project 2,
              //  EXCEPT that it throws a RuntimeException
              //  if n-th element does not exist
        }
        void printN(int n) {
              // Like firstN except that values are printed
              //  rather than formed into a list. After printing
              //  current line is terminated (like a println).
        }
}
```

To show that your lazy lists work compile and run class `Test` in `~cs538-1/public/java`. You'll see (again) that the sieve is slow. (Even slower than in ML!).

2. Once you have lazy lists working in Java, you can improve your implementation by using some of features of Pizza. In particular, you can make your lazy lists polymorphic, allowing, e.g., `LazyList<int>` or `LazyList<boolean>`.

   Pizza also allows first-class functions, so you can improve your implementation by using a member function that *is* a function (e.g.,

   `() -> LazyList f( ) { return seq(s,l);}`.

3. You are to redo question 4 of Assignment #2 (lazy lists and the "Sieve of Erastosthenes") in Python. Since Python is dynamically typed, you may represent null lazy lists as `[]` (just like ordinary Python lists). Moreover, a lazy list containing only one value may be represented as an ordinary list: `[ val ]`.
   Longer lazy lists will be represented as a list with two values: `[val, fct]`. `val` is the head of the list; it is an ordinary Python value. `fct` is a suspension function of no arguments; when called it will generate a lazy list representing the tail of the list.
   In Python function literals (lambda terms) are represented as

   ```
   lambda args: expression
   ```

   You must be careful though; Python allows access to locals and globals but not direct access to intermediately scoped identifiers. You can use the following "trick" to allow access to intermediately scoped identifiers in a lambda term.
   Python allows parameters with default values. If a parameter is not given a value at

the point of call, the default is used. The syntax (in a lambda term) is:

```
lambda id1=val1,...,idn=valn: expression
```

If $id_1$ to $id_n$ aren't given values in a call, the values of $val_1$ to $val_n$ will be available by using the names $id_1$ to $id_n$, and the values of $val_1$ to $val_n$ **can be** intermediately scoped identifiers. Thus in

```
def retFct(a):
    return lambda x=a:x
```

function `retFct` returns a function whose value, when called without parameters, is the value of `a`. You can use a similar approach when you build suspensions for lazy lists.

In ML the function `Nth` returns `None` if the n-th element of a list does not exist. In Python you can use the special value `None` for the same purpose.

Since Python is interpreted, you may find that the call `Nth(primes(),20)` takes too long. If so, try a simpler call like `Nth(primes(),19)` or `Nth(primes(),18)`.

## What to Hand In

Submit your solution electronically by placing your files in your handin directory: `~cs538-1/public/handin/proj4/your-login`. Each file should include a comment that contains

```
your name, your login
```

Be sure to make clear which of the three programs you have decided to implement.