Mid-Term Exam

May 1, 1997

Name: ______________________________

Good luck!

Problem 1 (10 points)

Write the Scheme function *applyList* that takes a list of functions and a number and returns a new list constructed by applying each of the functions to the number.

```
> (applyList (list add1 sub1 zero?) 7)
(8 6 #f)
```

Problem 2 (30 points)

The only thing one can do with a mathematical function (or a Scheme or C function for that matter) is to apply it. Thus, taking the object-oriented view, one can represent a function as an object with one method called `apply` that takes the argument to the function and returns the result. For example, if the object f represents the function `(lambda (x) (+ x 1))`, then the result of `f.apply(3)` is 4.

Furthermore, a family of related functions can be represented by a class. The constructor for the class takes a parameter, and creates an object representing a function from the family. For example, the family of functions:

```
(lambda (x) (+ x 1))
(lambda (x) (+ x 2))
(lambda (x) (+ x 3))
...
```

can be represent by a class `AddFunction` whose constructor takes a number i and returns an object representing the function `(lambda (x) (+ x i))`.

- Using the above idea, write the definition of the class `AddFunction`.

- Write a function (static method) `applyArray` that behaves like the Scheme function *applyList* from Problem 1. The arguments to the method should be an array of `AddFunction`s and a number. The result should be a new array containing the results of applying each function to the number. (The length of an array a can be obtained by `a.length`.)

Here is a small test that should print 8, 9, and 10:

```
public class Main {
  public static void main (String[] args) {
    AddFunction f1 = new AddFunction (1);
    AddFunction f2 = new AddFunction (2);
    AddFunction f3 = new AddFunction (3);
    AddFunction[] fs = new AddFunction[3]; // creates an array of 3 functions
    fs[0] = f1;
    fs[1] = f2;
    fs[2] = f3;
    int[] results = applyArray (fs, 7);
    for (int i=0; i<3; i++)
      System.out.println(results[i]);
  }
}
```

Problem 3 (10 points)

Write a Scheme function *merge* that merges two sorted lists of numbers into a larger sorted list. Assume the elements are ordered from smallest to largest.

```
> (merge '() '(1 2 3))
(1 2 3)
> (merge '(1 2 3) '())
(1 2 3)
> (merge '(1 3 5 7 8 9 10) '(2 4 6))
(1 2 3 4 5 6 7 8 9 10)
```

Problem 4 (10 points)

Write a Scheme function *split* that takes a list and splits in two sublists. The first sublist should contain the elements at odd indices, and the second sublist should contain the elements at even indices.

```
> (split '())
(() ())
> (split '(1))
((1) ())
> (split '(1 2 3 4 5 6 7 8 9))
((1 3 5 7 9) (2 4 6 8))
> (split '(1 2 3 4 5 6 7 8 9 10))
((1 3 5 7 9) (2 4 6 8 10))
```

Problem 5 (10 points)

Write a Scheme function *mergesort* that takes a list of numbers and returns a sorted version from smallest to largest. Use the functions *split* and *merge* above.

```
> (mergesort '())
()
> (mergesort '(8 1 3 9 6 5 7 2 4 10))
(1 2 3 4 5 6 7 8 9 10)
```

Problem 6 (30 points)

Convert the functions *merge*, *split*, and *mergesort* to continuation-passing style.