Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.826 *Principles of Computer Systems*

February 1, 2000 **Due: Tuesday, February 8, 2000**

Problem Set 1

Instructions: There are **six** *problems in this problem set; please turn in each problem on a separate sheet of paper. Also give the amount of time you spend on each problem.*

Problem 1

(a) Write an expression in *SPEC* whose value is the factorial function defined on the range $[0, \infty)$ and that is undefined outside that range.

(b) Let `fact` denote the value of the expression from Part 1. Write an expression, using a function constructor, whose value is a function that returns `fact` on the range $[0, \infty)$ and 0 at -1.

Problem 2

Given the following *SPEC* code:

```
TYPE S = SEQ STRING

VAR s0, s1, s2 |
  s0 := {‘‘pat’’, ‘‘sam’’, ‘‘leslie’’, ‘‘chris’’}
  s1 := {‘‘leslie’’, ‘‘leslie’’, ‘‘jean’’, ‘‘jean’’, ‘‘chris’’}
  s2 := {‘‘chris’’, ‘‘leslie’’}
```

(a) Give a sequence `p` such that `p == s0 - s1`.

(b) Is it always true that `s0 - s1 == s0 - s2`?

(c) Is it always true that `s0 - s1 = s0 - s2`?

Problem 3

A tree of integers has the *heap property* if the keys of all children of every node x are less than or equal to the key of node x. Assume that trees are represented with two functions, `parent`, of type `Int` $\rightarrow$ `Int`, and `children`, of type `Int` $\rightarrow$ `Set[Int]`.

(a) In *SPEC*, write a specification for a `heapify` function that takes as input a tree (represented by `parent` and `children` functions) and produces as output a tree whose root has the heap property.

(b) Also in *SPEC*, write an implementation of module you specified in Part 1.

Problem 4
For each of the following *SPEC* programs, list all of the possible final values of `x`.

(a)
```
<< VAR x: INT := 0, y: INT |
  IF y > 0 => x := 1
    [] y < 10 => x := 2
    [*] x := 3
  FI >>
```

(b)
```
<< VAR x: INT := 0, y: INT |
  IF y > 0 => x := 1
    [] y < 10 => x := 2
  FI >>
```

(c)
```
<< VAR x: INT := 0, y: INT |
  IF y > 0 => x := y + 1
    [] y < 0 => x := y - 1
  [*] x := y >>
```

Problem 5
An array partitioning routine takes an array of integers and partitions the array about one of its elements.

(a) What should the type of the partitioning routine be? (`FUNC`, `APROC`, or `PROC`?)

(b) Write a specification for the partitioning routine. Be careful not to over-constrain your specification.

(c) Write an implementation for the module you specified in Part 2. Your implementation should partition the array about a randomly-chosen element.

Problem 6
Consider a routine that takes a `REAL`-valued function `func`, a `REAL` parameter `tol`, and a starting guess `x_0`, that uses Newton's Method to find a zero of `func` with tolerance `tol`.

(a) What type should the routine be? (`FUNC`, `APROC`, or `PROC`?)

(b) Write a specification in *SPEC* of the routine.

Note that in some circumstances Newton's method may fail to converge. You should choose a subset of functions on which you are able to specify conditions for convergence, and write your specification accordingly.

(c) Write an implementation in *SPEC* of the module you specified in Part 2.