

Midterm Examination

Your **full** name:

UCSD id number:

No books are allowed, but you may use one double-sided sheet of notes in your own handwriting.

Look through the whole exam and answer the questions that you find easiest first.

Whenever you make an assumption, state it clearly. Whenever you are asked to give example code in a certain programming language, you are not required to get the syntax exactly right, but your code should be recognizable and understandable, with the most important language keywords used appropriately.

(Question 1) [20 points] A two-dimensional array can be represented in ML as a list of lists. For example, a 3 by 3 array of integers can be represented as `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`.

(a) [3 points] Write a recursive ML function that sums the components of a two-dimensional array of integers represented in this way.

(b) [4 points] Write a recursive ML function that sums the components of an array of integers with any number of dimensions that is represented by nested lists, i.e. as a list of lists of lists etc.

(c) [3 points] Pure ML does not have standard arrays. In addition to nested lists, explain an alternative way of representing an array in ML.

(d) [4 points] In numerical computing, assigning a new value to one component of an array is a very common operation. Given this fact, explain as precisely as possible why nested lists are *not* an efficient way of implementing arrays. Is the alternative you suggested in part (c) efficient?

(e) [3 points] Based on your answer to part (d), explain why functional programming languages like pure ML are not well-designed for numerical computing.

(f) [3 points] Professor Sussman of MIT has argued that functional PLs are good for numerical computing, because they provide higher-order functions. Based on your experience with the first CSE 130 project, explain Prof. Sussman's opinion.

(Question 2) [20 points] According to Dershem and Jipping (*Programming Languages: Structures and Models*, PWS Publishing Co., 1995, p. 42):

An *abstraction* is a representation of an object that includes only the relevant attributes of the original object, ignoring those attributes that are irrelevant to the purpose at hand. ... With data, the programmer is able to work more effectively by using simpler abstractions that do not include many irrelevant details of data objects. With procedures, abstractions facilitate good design practices and modularity.

- (a) [4 points] Consider a dictionary data structure implemented as a binary tree. Sketch a typical C representation for this data structure. Which attributes of the data structure are ignored by the C representation? Are these attributes always irrelevant?
- (b) [4 points] Sketch a recursive type representing the same data structure. Which attributes are ignored by this representation? Are these attributes always irrelevant?
- (c) [4 points] Sketch a pure abstract data type (ADT) representing the same data structure. Which attributes are ignored by this representation? Are these attributes always irrelevant?
- (d) [4 points] Explain these two claims:
 - (i) A procedure with static scoping is more abstract than the same procedure with dynamic scoping.
 - (ii) A pure function with no side-effects is more abstract than a subroutine with side-effects.
- (e) [4 points] Explain how “abstractions facilitate good design practices and modularity.”

This page is left blank to provide additional space for answers.