

# Programming Languages and Systems Comprehensive Exam

August 17, 1998  
Computer Science Department  
University of New Mexico

The test monitor will answer some parts of these questions for you if you are unsure of what some of these terms mean. The only penalty is that you will not get any points for the part that was answered for you. You can still answer the other parts of the question. This way can get you started on a question even if you are not sure of the definitions.

You can do any questions you want. You do not have to do all the parts of a single question. Instead you can pick the parts you want to answer. Be **sure** that it is absolutely clear which questions you are answering.

Do enough questions (or parts of questions) to total at least 100 points. It is okay to go over 100 points and you can answer as many questions as you wish but all the questions you answer will be counted in your grade and the total will be normalized to 100 points. We will **not** take the best 100 points.

## 1. **Lazy languages**

- (a) (5 points) Explain what is meant by a *lazy language*.
- (b) (7 points) Give a program in a lazy language that shows the lazy features of that language. (The language you use could just be Scheme with the assumption that it is a lazy version of Scheme.)
- (c) (8 points) Give an example of a problem that is more easily programmed in a lazy language than in other types of languages. This example should be different from the program in part (b).

## 2. **Using functions** (10 points) An array can be thought of as a function which takes the array index as its argument and returns the value at that index in the array. Show how to implement this literally in Scheme, that is, implement arrays as Scheme functions. The array will start out as a function which returns "index out of range" for every argument. You read from the array just by using the array as a function, for example: `(set x (arr1 7))`. You set a value in the array by using the old array function to define a new array function, for example: `(set arr1 (set-array-value arr1 index value))`.

## 3. **Prolog**

- (a) (6 points) "Prolog programs are a fairly close approximation of declarative specifications." Describe what this statement means in more detail.
- (b) (9 points) Because of (a), the Prolog interpreter may be viewed as a theorem prover. Specify in what sense this is true and in what sense this is not true. (That is, how does the procedural semantics of the prolog interpreter compare to that of a traditional theorem prover).

## 4. **Multiple Inheritance** Some languages allow multiple inheritance of interfaces and implementations.

- (a) (5 points) What problems does multiple inheritance create? What problems does multiple inheritance solve?
- (b) (5 points) Suppose we only allow multiple inheritance of pure virtual functions (say in C++). Which of the problems of multiple inheritance does this solve and which does it not solve?

- (c) (5 points) It has been said that all instance of multiple inheritance can be handled just as well as a *has-a* relation. Analyze this statement. In what ways it is true and in what ways it is false?
5. **Macros** Many languages have a macro facility. They were first used in assembly languages and helped to relieve some of the tedious and repetitive coding required by assembly language.
- (a) (4 points) What is the difference between a macro and a function?
- (b) (2 points) Briefly describe the macro facility in C++.
- (c) (4 points) Briefly describe the macro facility in Scheme or Common Lisp.
- (d) (4 points) Explain how the C++ template facility is a kind of macro facility. Explain why the C macro facility cannot be used to implement templates.
- (e) (4 points) Compare a macro invocation with a function call in a lazy language. How are they different and how are they the same?
6. **Threads**
- (a) (4 points) A *thread-safe* program works correctly with multiple threads operating simultaneously. What is the relationship (if any) between thread-safeness and mutual exclusion?
- (b) (4 points) Is thread-safeness a property of a program or a process? Explain.
- (c) (8 points) Describe two mechanisms intended to deal with the issue of mutual exclusion between threads. Use two mechanisms that have actually been included in some reasonably well known programming language. Describe each mechanism in terms of one language it was implemented in (and, of course, state what language it is). The two mechanisms should be basically different mechanisms, not the same mechanism implemented in two different languages.
7. **Closures**
- (a) (5 points) What is a closure? Give an example.
- (b) (15 points) C++ does not have closures but we can use classes and objects to simulate some of the functionality of closures. As an example, let us consider currying in a functional language like Scheme. Here is an example of the `curry` function and its use:
- ```
(define curry
  (lambda (f)
    (lambda (arg1)
      (lambda (arg2)
        (f arg1 arg2))))))
(define plus1 ((curry +) 1))
(plus1 5)
```
- You can get most of the effect of `curry` in C++ using objects. Show how this can be done. *Do not use function pointers* but you can use pointers to objects. You can overload the function call operator (operator `()`) or you can just define a member function named `call`.
- (c) (5 points) Explain what aspects of closures you can simulate with the method used in (b) and which you cannot.
8. **Design patterns and interfaces**
- (a) (13 points) An important concept is the idea of interfaces, that is, the only thing one object should know about another object is the interface it implements. Give examples from a few design patterns that exemplify the interface idea.
- (b) (7 points) Many objects will implement several interfaces but another object might only know one of the interfaces it implements. Explain why is useful to structure things this way. Give an example of a situation where this would happen.