

Programming Languages and Systems Comprehensive Exam

August 14, 2000
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 can get you started on a question even if you are not sure of the definitions.

You can do any of the numbered questions. You must do all parts of the questions you pick. Be **sure** that it is absolutely clear which questions you are answering.

Do enough 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. Static and dynamic** (20 points) Static versus dynamic is a common dichotomy in programming languages.
 - (5 points) Define what is meant, in general, by the terms static and dynamic in the context of programming languages. If your examples in part b imply two or more definitions, give all definitions that are used of these terms in programming languages.
 - (15 points) Give several different examples of the static versus dynamic dichotomy in specific programming languages or between pairs of programming languages. For each example, describe how a programming languages facility or feature can be defined to be either static or dynamic and the tradeoffs between them. Explain each example carefully. Try to come up with examples that cover as wide a range of concepts as possible. They should not be small variations on the same idea. Try to give examples from as wide a range of programming languages as possible. Also if these terms are used in several distinct ways in programming languages then try to give an example of each distinct definition. Be sure to distinguish between different definitions of static and dynamic and different examples of what is essentially the same definition. We want to be sure you understand the difference.
- 2. Composition versus inheritance** (20 points) In recent years many experts have been advising that you favor composition over inheritance in designing object-oriented systems.
 - (7 points) Explain what this means. What are composition and inheritance?
 - (13 points) Give an example where you can achieve a design goal using either composition or inheritance. Describe how each method can be used to solve the design problem. Compare the two solutions.
- 3. Portability** (30 points) Suppose our goal is to be able to run programs in a language on as many different computing platforms as possible. There are several possible approaches to this problem:
 - an interpreter
 - a compiler
 - a byte-code interpreter
 - machine language to machine language translation.In addition we could apply "just-in-time" translation to some of these approaches.

Compare these various approaches to portability. Do not ramble in your answer and do not answer with only text. Use tables (with footnotes when necessary). Be organized. You have to decide the factors you will use to compare them. Here are some things to consider although you might consider other factors as well.

- (1) Where (on the source platform or the target platform) and when parsing is done.
- (2) Where and when code is generated.

(3) What things must be done on the source platform and what things must be done on the target platform and what things can be done on either platform.

(4) Where optimization occurs.

(5) What tools (translators, optimizers, debuggers) are necessary (or desirable) and when and where they will run. For the tools you should consider whether you need only one version, or one for each source, or one for each target, or something else.

(6) Do any of these forms of portability depend on each other? For example does a portable interpreter require a portable compiler?

4. **Lisp s-expressions** (10 points) The symbolic expressions (s-expressions) in Scheme and Lisp have been called universal data structures because they can be used to implement any kind of data structure (although maybe not efficiently). Show how to implement the following data structures with s-expressions: array, C structure, tree, map (also called a dictionary or associative array), and graph.
5. **Exceptions** (20 points)
 - a) (5 points) Most exception mechanisms use a "block-termination" policy with exceptions, that is, the block that the exception occurred in is always terminated and cannot be restarted. A few languages allow the exception handler to try to fix things up and retry the operation that got the exception. This policy is not common. Explain why not. What are the problems with this policy.
 - b) (5 points) Besides, the user can program retries with most exception handling mechanisms. Show how this would be done in C++ (or another language with an exception facility).
 - c) (5 points) Suppose you have a programming language that has an exception handling mechanism and also supports threads. Can you think of any problems the interactions between these two mechanisms might cause? Consider both the precise specification of the semantics of the language and the implementation of the language.
 - d) (5 points) In C++, declarations are really executable statements. Explain why this is so and why this is a significant change from C. Declarations can occur anywhere inside a block, not just at the beginning. What problems can this cause for an exception handler?
6. **Object-Oriented I/O Systems** (20 points) Suppose you were designing the I/O system for an object-oriented language. The I/O system should support the following features:
 - Input from and output to files, pipes, and in-memory strings.
 - Input from and output in terms of bytes only or any of the basic types of the language.
 - Input and output in printable form.
 - Optional buffering on any input or output.
 - Random access to files (read and write) should also be allowed.Design a set of classes that implement these facilities. Just specify the classes and their operations; you do not need to do any implementation. Naturally, we would expect some inheritance to be used.
7. **Garbage collection** (20 points) Discuss the tradeoffs between sweep-and-mark and stop-and-copy garbage collection. What is a conservative garbage collector? When is each one appropriate?