

Programming Languages and Systems Comprehensive Exam

January 8, 2001
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.

Do any five questions. You must do all parts of the questions you pick. Be sure to turn in *only* five questions.

1. **Abstraction mechanisms** (20 points) Two useful reuse mechanisms are parameterized types (called templates in C++) and inheritance. Suppose you wanted to define a stack (using a linked list for implementation) that could either be a stack of integers or a stack of floats.
 - a) (6 points) Show how you would do this using parameterized types. You can do this using with C++ templates or in any other language that has parameterized types. Show the generic definition and show how to define a stack of integers and a stack of floats using the generic definition. The generic definition should allow the user to define stacks of any data type.
 - b) (6 points) Do the same thing you did in part (a) using inheritance.
 - c) (8 points) Compare these two solutions in terms of clarity, ease of writing, time and space efficiency, generality, ease of extension (to more types than just integer and float), type safety, static versus dynamic issues, and any other criteria you think are important.
2. **Byte-code Compilers** (20 points)
 - a) (6 points) How does a byte-code compiler improve the portability of a programming language?
 - b) (6 points) Byte-codes can be further compiled into machine code. What are the advantages of doing the compilation in two steps like this? What are the disadvantages?
 - c) (8 points) What is meant by *just-in-time compilation* (also know as *on-the-fly compilation*)? What are the advantages and disadvantages of it? Does just-in-time compilation apply only to byte-code compilation or could it be used in other contexts? Explain why not or give an example of another place it could be used.
3. **Continuation-passing** (20 points) What is the continuation-passing style (CPS)? What does it make explicit? Convert this code to CPS: (define (factorial n ...)) (Fill in the usual code for factorial.)
4. **C++ STL function objects (20 points) Consider the following C++ code.**

```
class LessThan {
public:
    LessThan(int n) : _n(n) {}
    bool operator() (const int a) { return a < n; }
};

vector<int> v;
// fill v by some means
vector<int>::iterator v_end;
v_end = remove_if(v.begin(), v.end(), LessThan(6));
```

The “remove_if” function goes through the v vector and removes items that pass the test defined by the third argument.

- a) (8 points) Explain how the above example works, that is, explain what is happening in the code. You will need to speculate how remove_if is implemented.
 - b) (6 points) Show how this would be implemented in Scheme or Common Lisp.
 - c) (6 points) Compare the solutions in C++ and in Scheme or Common Lisp. How are they similar and how are they different? Which is more powerful, or are they not comparable?
5. **Scheme** (20 points)
- a) (5 points) A lambda expression is an important generalization of Lisp and Scheme functionality. Describe what this means in more detail.
 - b) (15 points) Create a (small) set of lambda expressions that implements an object-oriented simulation. In particular, show how lambdas can be used to build in messages and inheritance.
6. **ML** (20 points) The SML/NJ compiler, confronted with a user program that is not type-correct, often responds with a long listing of errors, from which it is difficult for the user to figure out what is wrong with the program. Describe how the ML compiler determines type-correctness, and discuss why it is inherently impossible always to provide error messages consistent with a programmer's intuitive notion of the source of the error.
7. **Translation** (20 points) You are given the code for a stack machine. The instruction set includes instructions *push a*, *pop a*, where *a* specifies a memory address, as well as *add*, *mul*, etc. (various arithmetic operations). There are no control instructions. Write (in your favorite programming language) a translator to translate this straight-line code into code for a RISC machine. (Don't waste time on parsing; assume the code is already in a suitable internal form.) What is involved in producing efficient RISC code as the output?