Review Worksheet

Spring 2002

If you can answer all the following questions, you should have no problem with the final.

1. What is the difference between concrete and abstract syntax?
2. Is there a 1-1 correspondence between concrete and abstract syntaxes?
3. How can a grammar specify precedence and associativity?
4. What is an ambiguous grammar? Why are ambiguous grammars OK for abstract but not concrete syntax?
5. How many general uses can you name for higher-order functions?
6. Why might one curry the arguments of a function?
7. What is lexing, and what is parsing?
8. How many tokens are there likely to be in the code `fn x:int => fn y:int => x+y` ? What might its abstract syntax tree look like?
9. What is an $LL(1)$ grammar?
10. What's a recursive-decent parser? How is it constructed?
11. What's the difference between an interpreter and a compiler?
12. What is lexical scoping? Dynamic scoping? Could a language be neither?
13. Can you read and verbally explain the inference rules we looked at?
14. Can you use those inference rules to do a simple proof?
15. Could you complete the following implementation of the `Metavar` structure ?

```
structure Metavar =
  struct
     type 'a metavar = ('a option) ref
     fun new():'a metavar = ...
     fun get(m:'a metavar):'a = ...
     fun set(m:'a metavar,x:'a):unit = ...
  end
```

16. How can one tell if two assignable variables alias?
17. What does the subtyping relation mean (and which direction does it go) ?
18. What is the subsumption rule?
19. Given two types, how do you figure out which way the subtyping relation should naturally go?
20. What's the difference between overloading a method in a subclass, and overriding the method in a subclass? Can a single subclass do both to the same method of the superclass?
21. What's the difference between subtyping and inheritance?
22. What's a function pointer? What's a closure? Why is a function pointer less useful than a closure or an object? How do C programs typically get around this limitation?
23. What's the difference between call-by-{value,name,need,reference}?
24. What are the parenthesization conventions for λ-calculus?
25. What's α-equivalence?
26. What is capture-avoiding substitution?
27. What are Church Numerals?
28. What's the point of the Y combinator?
29. Why is it reasonable to show a λ-calculus expression is convertible to `tt` and claim we're programming, if only β-reduction (and not β-expansion) makes obvious sense computationally?
30. What things are isomorphic according to the Curry-Howard Isomorphism?
31. How does (monomorphic) type inference generate and solve typing constraints?
32. How do garbage collectors decide when data is dead? Why is this not optimal?