Type Systems

1. What purpose(s) do types serve in a programming language?

2. What does it mean for a language to be strongly typed? Statically typed? What prevents, say, C from being strongly typed?

3. Name two programming laguages that are strongly but dynamically typed.

4. What is a type clash?

5. Discuss the differences among the denotational, structural, and abstraction-based views of types.

6. What does it mean for a set of language features (e.g., a type system) to be orthogonal?

7. What are aggregates?

8. What are option types? What purpose do they serve?

9. What is polymorphism? What distinguishes its parametric and subtype varieties? What are generics?

10. What is the difference between discrete and scalar types?

11. Give two examples of languages that lack a Boolean type. What do they use instead?

12. In what ways may an enumeration type be preferable to a collection of named constants? In what ways may a subrange type be preferable to its base type? In what ways may a string be preferable to an array of characters?

13. What is the difference between type equivalence and type compatibility?

14. Discuss the comparative advantages of structural and name equivalence for types. Name three languages that use each approach.

15. Explain the difference between strict and loose name equivalence.

16. Explain the distinction between derived types and subtypes in Ada.

17. Explain the differences among type conversion, type coercion, and nonconverting type casts.

18. Summarize the arguments for and against coercion.

19. Under what circumstances does a type conversion require a run-time check?

20. What purpose is served by universal reference types?

21. What is a type inference? Describe three contexts in which it occurs.

22. Under what circumstances does an ML compiler announce a type clash?

23. Explain how the type inference of ML leads naturally to polymorphism.

24. Why do ML programmers often declare the types of variables, even when they don't have to?

25. What is unification? What is its role in ML?

26. Explain the distinction between implicit and explicit parametric polymorphism. What are their comparative advantages?

27. What is duck typing? What is its connection to polymorphism? In what languages does it appear?

28. Explain the distinction between overloading and generics. Why is the former sometimes called ad hoc polymorphism?

29. What is the principal purpose of generics? In what sense do generics serve to a broader purpose in C++ and Ada than they dp in Java and C#?

30. Under what circumstances can a language implementation share code amonf separate instances of a generic?

31. What are container classes? What do they have to do with generics?

32. What does it mean for a generic parameter to be constrained? Explain the difference between explicity and implicit constraints. Describe how interface classes can be used to specify constraints in Java and C#/

33. Why will C# accept `int` as a generic argument, but Java won't?

34. Under what circumstances will C++ instantiate a generic function implicitly?

35. Why is equality testing more subtle than it first appears?

36. Why is it difficult to produce high-quality error messages for misuses of C++ templates?

37. What is the purpose of explicit instantiation in C++? What is the purpose of `extern` templates?

38. What is template metaprogramming?

39. Explain the difference between upper bounds and lower bounds in Java type constraints. Which of these does C# support?

40. What is typeerasure? Why is it used in Java?

41. Under what circumstances will a Java compiler issue an "unchecked" generic warning?

42. Why must fields of generic parameter type be explicitly initialized in C#?

43. For what two main reasons are C# generics often more efficient than comparable code in Java?

44. Summarize the notations of covariance and contravariance in generic types.

45. Hoes does a C# delegate differ from an interface with a single method (e.g., the C++ chooser of Figure c-7.5)? How does it differ from a function pointer in C?