Composite Types

1. What are struct tags in C? How are they related to type names? How did they change in C++?

2. How do the records of ML differ from those of most other languages?

3. Discuss the significance of "holes" in records. Why do they arise? What problems do they cause?

4. Why is it easier to implement assignment than comparison for records?

5. What is packing? What are its advantages and disadvantages?

6. Why might a compiler reorder the fields of a record? What problems might this cause?

7. Briefly describe two purposes for unions/variant records.

8. What is an array slice? For what purpose are slices useful?

9. Is there any significant difference between a two-dimensional array and an array of one-dimensional arrays?

10. What is the shape of an array?

11. What is a dope vector? What purpose does it serve?

12. Under whatcircumstances can an array declared within a subroutine be allocated in the stack? Under what circumstances must it be allocated in the heap?

13. What is a conformant array?

14. Discuss the comparative advantages of contiguous and row-pointer layout for arrays.

15. Explain the difference between row-major and column-major layout for contiguously allocated arrays. Why does a programmer need to know which layout the compiler uses? Why doo mosy languages designers consider row-major layout to be better?

16. How much of the work of computing the address of an element of an array can be performed at compile time? How much must be performed at run time?

17. Name three languages that provide particularly extensive support for character strings.

18. Why might a language permit operations on strings that it does not provide for arrays?

19. What are the strengths and weaknesses of the bit-vector representation for sets? How else might sets be implemented?

20. Discuss the tradeoffs between pointers and the recursive types that arise naturally in a language with a reference model of variables.

21. Summarise the ways in which one dereferences a pointer in various programming languages.

22. What is the difference between a pointer and an address? Betwen a pointer and a reference?

23. Discuss the advantages and disadvantages of the interoperability of pointers and arrays in C.

24. Under what circumstances must the bounds of a C array be specified in its declaration?

25. What are dangling references? How are they created, and why are they a problem?

26. What is garbage? How is it created, and why is it a problem? Discuss the comparative advantages of reference counts and tracing collection as a means of solving the problem.

27. What are smart pointers? What purpose do they serve?

28. Summarize the differences among mark-and-sweep, stop-and-copy, and generational garbage collection.

29. What is pointer reversal? What problem does it address?

30. What is "conservative" garbage collection? How does it work?

31. Do danging references and garbage ever arise in the same programming language? Why or why not?

32. Why was automatic garbage collection so slow to be adopted by imperative programming languages?

33. What are the advantages and disadvantages of allowing pointers to refer to objects that do not lie in the heap?

34. Why are lists so heavily used in functional programming languages?

35. What are list comprehensions? What languages support them?

36. Compare and contrast the support for lists in ML- and Lisp-family laguages.

37. Explain the distinction between interactive and file-based `I/O`; between temporary and persistent files.

38. What are some of the tradeoffs between supporting `I/O` in the language proper versus supporting it in libraries?

39. What are anonymous unions and structs? What purpose do they serve? How is this related to the integration of variants with records in Pascale and its descendants?

40. What is a tag (discriminant) in a variant record? In a language like Ada or OCaml, how does it differ from an ordinary field?

41. Discuss the type safety problems that arise with variant records. How can these problems be addressed?

42. Summarize the rules that prevent access to inappropriate fields of variant records in OCaml and Ada.

43. Why might one wish to constrain a variable, so that it can hold only one variant of a type?

44. Explain how classes and inheritance can be used to obtain the effect of constrained variant records.

45. What are tombstones? What changes do they require in the code to allocate and deallocate memory, and to assign and dereference pointers?

46. Explain how tombstones can be used to support compaction.

47. What are locks and keys? What changes do they require in the code to allocate and deallocate memory, and to assign and dereference pointers?

48. Explain why the protection afforded by locks and keys is only probabilistic.

49. Discuss the comparative advantages of tombstones and locks and keys as a means of catching dangling references.

50. Explain the differences between interactive and file-based `I/O`, between temporary and persistent files, and between binary and text files.

51. What are the comparative advantages of text and binary files?

52. Describe the end-of-line conventions of Unix, Windows, and Macintosh files.

53. What are the advantages and disadvantages of building `I/O` into a programming language, as opposed to providing it through a library routines?

54. Summarize the different approaches to text `I/O` adopted by Fortran, Ada, C, and C++.

55. Describe some of the weaknessess of C's `scanf` mechanism.

56. What are stream manipulators? How are they used in C++?