

7 Data Types

7.12 Exercises

- 7.27 How would you implement the final version of our `element` type in ML? How would you extract the fields of the variant part? Specifically, suppose you have declared a record to represent copper. How would you specify the equivalent of `copper.source`?
- 7.28 In Example 6.67 we described a programming idiom in which an iterator takes a “loop body” function as argument, and applies it to every element of a given container or set. Show how to use this idiom in ML to apply a function to every element of the tree in Example ©7.113. Write versions of your iterator for preorder, inorder, and postorder traversals.
- 7.29 Rewrite the left half of Example ©7.121 in C++ using *references* (see Section 8.3.1).
- 7.30 Show how variant records in Pascal or unions in C can be used to interpret the bits of a value of one type as if they represented a value of some other type. Explain why the same technique does not work in Ada. After consulting an Ada manual, describe how an unchecked pragma can be used to get around the Ada rules.
- 7.31 Are variant records a form of polymorphism? Why or why not?
- 7.32 Pascal does not permit the tag field of a variant record to be passed to a subroutine by reference (i.e., as a `var` parameter). Why not?
- 7.33 Explain how to implement dynamic semantic checks to catch references to uninitialized fields of a tagged variant record in Pascal. Changing the value of the tag field should cause all fields of the variant part of the record to become uninitialized. Suppose you want to avoid adding flag fields within the record itself (e.g., to avoid changing the offsets of fields in a systems program). How much harder is your task?
- 7.34 Explain how to implement dynamic semantic checks to catch references to uninitialized fields of an *untagged* variant record in Pascal. Any assignment to a field of a variant should cause all fields of other variants to become

uninitialized. Any assignment that changes the record from one variant to another should also cause all other fields of the new variant to be uninitialized. Again, suppose you want to avoid adding flag fields within the untagged record itself. How much harder is your task?

- 7.35 We noted in Section ©7.3.4 that Pascal and Ada require the variant portions of a record to occur at the end, to save space when a particular record is constrained to have a comparatively small variant part. Could a compiler rearrange fields to achieve the same effect, without the restriction on the declaration order of fields? Why or why not?
- 7.36 In Example 7.88 we noted that reference counts can be used to reclaim tombstones, failing only when the programmer neglects to manually delete the object to which a tombstone refers. Explain how to leverage this observation to catch memory leaks at run time. Does your solution work in all cases? Explain.
- 7.37 Learn about the *smart pointer* design pattern in C++. Create a smart pointer package that uses tombstones to catch dangling references, and reference counts to reclaim garbage tombstones. Augment your package, as suggested in the preceding exercise, to catch (most) memory leaks.
- 7.38 Rewrite Example ©7.146 using `fgets`, `strtol`, `strtod`, etc. (read the man pages), so that it is guaranteed not to result in buffer overflow.
- 7.39 The `readln` and `writeln` procedures of Pascal give special treatment to ends of lines. By contrast, C's `printf` and `scanf` do not; they treat newlines and carriage returns like any other character. What are the comparative advantages of these approaches? Which do you prefer? Why?