Programming Languages

Spring 1993

1 hour

Answer four of the following six questions. When appropriate, you should take some time to explain your interpretation of the question. Since you have an hour, there should be time to explore each of the questions you choose in some depth (but please do not ramble on unnecessarily).

1. Describe subtyping and polymorphism. Explain the programming advantages and implementation costs of each.

2. Name three significant differences between Smalltalk and C^{++}. Describe a hypothetical or actual programming project and explain clearly why you would chose one of these languages over the other for this project. (Ignore differences in availability of implementations.)

3. Prolog is related to first-order logic, using a special form of first-order sentences. Specifically, a *Horn clause* is a sentence of the form

   $$(\forall x_1 \ \ldots \forall x_n)[C_1 \vee \ldots \vee C_k \longrightarrow D_1 \wedge \ldots \wedge D_\ell]$$

   In Prolog, this Horn clause may be written as the sequence

   $$\begin{array}{lll} D_1 & \text{:-} & C_1, \ldots, C_k \\ D_2 & \text{:-} & C_1, \ldots, C_k \\ \ldots & & \ldots \\ D_\ell & \text{:-} & C_1, \ldots, C_k \end{array}$$

   Page 174 of Rich and Knight, a textbook used in AI courses, contains the following paragraph about Prolog and its relationship to logic:

   > The fact that Prolog programs are composed only of Horn clauses and not of arbitrary logical expressions has two important consequences. The first is that because of the uniform representation a simple and efficient compiler can be written. The second consequence is even more important. The logic of Horn clause systems is decidable (unlike that of full first-order predicate logic).

   The second point is not clearly stated. What do you think the sentence means? (What do they think is decidable about Prolog?) Do you think the claim is correct? What implication would it have for Prolog programming?

4. What is the difference between a closure and an object? How are they similar? Does this mean you can do object-oriented programming in a language that allows higher-order functions? Higher-order functional programming with objects?

5. Suppose you wish to write part of a program in Lisp or Scheme, and half in Pascal. What sort of problems would you expect to encounter in passing values from one part of the program to another? Explain how these might be solved.

6. Someone suggests eliminating the run-time stack in a compiler. Instead of the stack, activation records will be allocated by the same mechanism as other dynamic allocation, and collected automatically be a garbage collector. In what situations, and for what languages, might this be a reasonable idea? When is it likely to be a losing proposition?