Handout #11 CS 342 — Principles of Programming Languages March 1, 1989

Homework 7: Ada

This homework is due Wednesday, March 22. Each weekday that the homework is late you lose 10% of the possible points.

1. (20 points) Design and implement an Ada package called `TYPE_DEF`, which exports a type that can be used to represent Ada type definitions (i.e., type expressions) and operations that can be used to manipulate them. You should describe your representation in general terms, and include an Ada package interface specification and body with comments. Design your type so that it would be useful in a tool that processes Ada programs (e.g., a compiler or syntax-directed editor) and so that it would be useful in the next problem.

   The following is a (simplified!) EBNF for type defintions in Ada.

   ⟨type definition⟩ ::=⟨enumeration type def⟩ | ⟨integer type def⟩ | ⟨real type def⟩
       | ⟨array type def⟩ | ⟨record type def⟩ | ⟨access type def⟩

   ⟨enumeration type def⟩ ::=(⟨enumeration literal list⟩)
   ⟨enumeration literal list⟩ ::=⟨enumeration literal⟩
       | ⟨enumeration literal⟩ , ⟨enumeration literal list⟩
   ⟨enumeration literal⟩ ::=⟨identifier⟩ | ⟨character literal⟩

   ⟨integer type def⟩ ::=⟨range constraint⟩
   ⟨range constraint⟩ ::=**range** ⟨range⟩
   ⟨range⟩ ::=⟨simple expression⟩ .. ⟨simple expression⟩

   ⟨real type def⟩ ::=⟨floating point constraint⟩
   ⟨floating point constraint⟩ ::=⟨floating accuracy def⟩
       | ⟨floating accuracy def⟩ ⟨range constraint⟩
   ⟨floating accuracy def⟩ ::=**digits** ⟨simple expression⟩

   ⟨array type def⟩ ::=⟨unconstrained array def⟩ | ⟨constrained array def⟩
   ⟨unconstrained array def⟩ ::=**array** (⟨index subtype list⟩) **of** ⟨subtype indication⟩
   ⟨constrained array def⟩ ::=**array** (⟨index constraint⟩) **of** ⟨subtype indication⟩
   ⟨index subtype list⟩ ::=⟨index subtype def⟩ | ⟨index subtype def⟩ , ⟨index subtype list⟩
   ⟨index subtype def⟩ ::=⟨type mark⟩ **range** <>
   ⟨index constraint⟩ ::=(⟨discrete range list⟩)
   ⟨discrete range list⟩ ::=⟨discrete range⟩ | ⟨discrete range⟩ , ⟨discrete range list⟩
   ⟨discrete range⟩ ::=⟨subtype indication⟩ | ⟨range⟩

   ⟨record type def⟩ ::=**record** ⟨component list⟩ **end record**
   ⟨component list⟩ ::=⟨component declaration⟩ ⟨component declaration⟩*
       | ⟨component declaration⟩* ⟨variant part⟩ | **null** ;
   ⟨component declaration⟩ ::=⟨identifier list⟩ : ⟨subtype indication⟩ ;
   ⟨variant part⟩ ::=**case** ⟨simple name⟩ **is** ⟨variant⟩ ⟨variant⟩* **end case** ;

⟨variant⟩ ::=**when** ⟨choice list⟩ => ⟨component list⟩
⟨choice list⟩ ::=⟨choice⟩ | ⟨choice⟩ | ⟨choice list⟩
⟨choice⟩ ::=⟨simple expression⟩ | ⟨discrete range⟩ | **others** | ⟨simple name⟩
⟨simple name⟩ ::=⟨identifier⟩

⟨access type def⟩ ::=**access** ⟨subtype indication⟩

⟨subtype indication⟩ ::=⟨type mark⟩ | ⟨type mark⟩ ⟨constraint⟩
⟨constraint⟩ ::=⟨range constraint⟩ | ⟨floating point constraint⟩
⟨type mark⟩ ::=⟨identifier⟩

Some notes on the grammar follow. A ⟨subtype indication⟩ is the name of a type, which (for purposes of this problem set) should be considered to be unstructured. The lexical syntax of identifiers, character literals, and is unimportant for this problem and not given here. A ⟨simple expression⟩ is just an expression used to denote an integer in most cases above. The built-in types **boolean**, **integer**, and so on are predeclared names for various types defined above. For example, **boolean** is a predeclared enumeration type and **integer** is a predeclared integer type.

2. This problem explores simulating Ada packages in Pascal and structural type equivalence.
   (a) (10 points) Translate your Ada package from the previous problem into a series of Pascal type definitions and some functions and procedures that operate on them. Call the type that represents Ada type definitions `type_def`.
   (b) (5 points) Define (in English) when an expression of type `E` is "assignment compatabile" with a variable of type `V`. Your rule should be as permissive as possible, but consistent with the security principle. For example, if `E` and `V` are both record types with the same field names and corresponding types, you should allow the assignment. Briefly justify your rule.
   (c) (10 points) Write a Pascal function `can_assign` that compares two `type_def`s for assignment compatability according to your rule in part (b). Use the type definitions and functions from part (a). Mail your program source code, and turn in listings and test data that shows that your program correctly implements the rule of part (b).
3. (3 points) Are linguistic mechanisms needed to enforce information hiding? Justify your answer briefly, using your experience from the previous problem.
4. (3 points) Explain how Ada's exception handling mechanism violates the structure principle.
5. (4 points) Write an Ada subprogram whose results differ depending on whether a given composite parameter is passed by reference or by copying (i.e., by value-result).
6. (5 points) A program that depends on the implementation of Ada, such as the one in the previous exercise, is not a legal Ada program. However, compilers are not required to detect such situations because doing so may be arbitrarily difficult. Discuss this philosophy of defining errors that cannot be easily checked by a compiler. Describe any alternatives to this philosophy and briefly compare the alternatives to the Ada designer's philosophy.