316 12-01-09

Note Title 12/1/2009

Final = 12-17-09 @ 6:15-8:15

Topics = Dynamic Semantics → Prolog

Review: Notes, Exercise Sets 3-9, Course Notes 7-17

- Static Variables
- Stack-Dynamic Variables
- Heap-Dynamic Variables
  - Explicit Heap-Dynamic Variables - use of explicit constructor functions (C++/C, Java)
  - Implicit Heap-Dynamic Variables

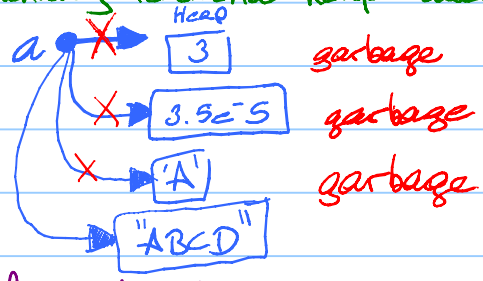
Statically-Typed Languages Vs. Type-Free Languages

- Statically-Typed
  - Languages in which types of all names get bound statically, mostly at program writing time
  - Type declarations in programs
  - PASCAL, ADA, C++, Java, Eiffel, C#, etc.
  - The compiler can detect all (or almost all) type errors
    → Type safety - Sometimes known as "strongly typed"
  - Promotes a disciplined style of programming based on clear organization of data and objects structured by types.
  - Generally leads to more efficient implementations
    - The compiler can allocate memory cells of fixed size and structure, according to the variable's type
    - Memory cells don't have to carry type descriptors
      - one exception = class objects in the heap
- Type-Free
  - Variables can be assigned values of any type at any time
  - No type declarations in programs
  - JavaScript, SmallTalk, LISP, PROLOG

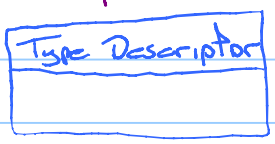
- Data structures can contain mixed-type data/objects
- Generic, polymorphic functions readily written
- No type safety
  - Type errors may become latent for a long time

} "Free-Wheeling" style of programming

- Memory-cell allocation is fundamentally reference-heap based
  - example: `a = 3;` `a = 3.5e-5;` `a = 'A';` `a = "ABCD";`



- The runtime system must handle automatic heap cell allocation, depending on types of values being assigned → runtime overhead
- Heap cells have to carry type descriptors
- example: `a = a + 5;`



what's the type of this value?

- Type descriptors may have to be checked for operator applications

- Scope
- Almost all modern languages are statically scoped

[Statically Scoped - Scopes of names bound at program writing time]

- Normally, scopes are determined by program units like blocks, functions, classes, modules (packages)
- Also by explicit scope indicators (private, protected, public)
- Dynamic Scoping
  - Scopes of names depend on dynamic processes during runtime (eg = chain of function calls)
  - ex: main → A → B → C → D

    main → A → C → D
  - This makes it hard to determine scopes

- Overview of Lisp and Prolog
  - Commonalities
    - Mainly used in AI applications
      - ex: symbolic computation - mathematical software, automatic equation solving, formula simplification, automatic differentiation/integration
      - ex: natural language processing
      - ex: robots
      - ex = expert systems, knowledge-based systems - weather report assistance, financial market analysis
    - Very High-Level languages that permit manipulation of high-level symbolic data without worrying about low-level data structures
    - Prototyping
      - Use Lisp/Prolog for rapid prototyping
        → re-implement in conventional layers
    - Both are type-free
    - No static variables
      - All variables are stack-dynamic or heap-dynamic
    - Good garbage collectors needed
  - Differences
    - Lisp
      - Hybrid of procedural and functional languages → "Pure Lisp"
        - Assignments
        - Loops
    - Prolog
      - Example of logic language
        - program = set of logical statements $(A \leftarrow B_1 \wedge \ldots \wedge B_n)$
        - computation = logical deduction process
      - Main data structures
        - Terms = look like mathematical function applications $f(x, g(y,z), h(x,y,z))$