Chapter 7:: Data Types

Programming Language Pragmatics

Michael L. Scott

Copyright © 2005 Elsevier

Administrative Notes

- Mid-Term Test
  - Thursday, July 27 2006 at 11:30am
  - No lecture before or after the mid-term test
  - You are responsible for material presented in the lectures not necessarily covered in the textbook

Copyright © 2005 Elsevier

Administrative Notes

- Mid-Term Test (cont.)
  - Will cover chapters1-6
    - Chapter 1: complete
    - Chapter 2: chapter introduction and Section 2.1
    - Chapter 3: entire chapter except Section 3.5.1, 3.5.2, 3.3.4 and 3.3.5
    - Chapter 5: entire chapter
    - Chapter 6: chapter introduction & Section 6.1 (complete)

Copyright © 2005 Elsevier

Administrative Notes

- Assignment One
  - Due date: Friday, July 28 2006 at 1:00pm
  - Submit your assignment in the drop-box located at the Computer Science and Engineering undergraduate office
  - Late assignments are subject to a penalty of 10% each day
  - I may choose to mark only a subset of the assigned questions
  - You must "show your work" where appropriate to obtain full marks

Copyright © 2005 Elsevier

Administrative Details

- Drop Deadline
  - Computer Science Department drop-deadline for the course is July 31 2006
  - If you wish to drop the course before this deadline will have to petition to "drop late" as far as the registrars office is concerned, the drop deadline has passed (they treated the course as D2 - see registrars office)
    - Of course, the petition will have the support of the department and of myself as well and will be approved under these circumstances

Copyright © 2005 Elsevier

Review

- What are several different ways selection can be achieved ?
- What is the significance of short-circuiting with respect to selection ?
- Why are “case” constructs important e.g., why not simply use nested if statements ?
- What is a “jump table” ?
- Why is iteration important ?
- What are some issues we must be aware of with respect to counted loops ?

Copyright © 2005 Elsevier

Review

- What is an iterator ?
- Can a counted loop always be written as a conditional loop ?
- What is recursion ?
- What is required in order for recursion to work ?
- What is tail recursion ?
- What is the significance of tail recursion ?

Copyright © 2005 Elsevier

Data Types

- We all have developed an intuitive notion of what types are; what's behind the intuition?
  - collection of values from a "domain" (the denotational approach)
  - internal structure of a bunch of data, described down to the level of a small set of fundamental types (the structural approach)
  - equivalence class of objects (the implementor's approach)
  - collection of well-defined operations that can be applied to objects of that type (the abstraction approach)

Copyright © 2005 Elsevier

Data Types

- What are types good for?
  - implicit context
  - checking - make sure that certain meaningless operations do not occur
    - type checking cannot prevent all meaningless operations
    - It catches enough of them to be useful
- Polymorphism results when the compiler finds that it doesn't need to know certain things

Copyright © 2005 Elsevier

Data Types

- STRONG TYPING has become a popular buzz-word
  - like *structured programming*
  - informally, it means that the language prevents you from applying an operation to data on which it is not appropriate
- STATIC TYPING means that the compiler can do all the checking at compile time

Copyright © 2005 Elsevier

Type Systems

- Examples
  - Common Lisp is strongly typed, but not statically typed
  - Ada is statically typed
  - Pascal is almost statically typed
  - Java is strongly typed, with a non-trivial mix of things that can be checked statically and things that have to be checked dynamically

Copyright © 2005 Elsevier

Type Systems

- Common terms:
  - discrete types – countable
    - integer
    - boolean
    - char
    - enumeration
    - subrange
  - Scalar types - one-dimensional
    - discrete
    - real

Copyright © 2005 Elsevier

Type Systems

- Composite types:
  - records (unions)
  - arrays
    - strings
  - sets
  - pointers
  - lists
  - files

Copyright © 2005 Elsevier

Type Systems

- ORTHOGONALITY is a useful goal in the design of a language, particularly its type system
  - A collection of features is orthogonal if there are no restrictions on the ways in which the features can be combined (analogy to vectors)

Copyright © 2005 Elsevier

Type Systems

- For example
  - Pascal is more orthogonal than Fortran, (because it allows arrays of anything, for instance), but it does not permit variant records as arbitrary fields of other records (for instance)
- Orthogonality is nice primarily because it makes a language easy to understand, easy to use, and easy to reason about

Copyright © 2005 Elsevier

Type Checking

- A TYPE SYSTEM has rules for
  - type equivalence (when are the types of two values the same?)
  - type compatibility (when can a value of type A be used in a context that expects type B?)
  - type inference (what is the type of an expression, given the types of the operands?)

Copyright © 2005 Elsevier

Type Checking

- Type compatibility / type equivalence
  - Compatibility is the more useful concept, because it tells you what you can DO
  - The terms are often (incorrectly, but we do it too) used interchangeably.

Copyright © 2005 Elsevier

Type Checking

- Certainly format does not matter:

```
struct { int a, b; }
```

is the same as

```
struct {
    int a, b;
}
```

We certainly want them to be the same as

```
struct {
    int a;
    int b;
}
```

Copyright © 2005 Elsevier

Type Checking

- Two major approaches: structural equivalence and name equivalence
  - Name equivalence is based on declarations
  - Structural equivalence is based on some notion of meaning behind those declarations
  - Name equivalence is more fashionable these days

Copyright © 2005 Elsevier

Type Checking

- There are at least two common variants on name equivalence
  - The differences between all these approaches boils down to where you draw the line between important and unimportant differences between type descriptions
  - In all three schemes described in the book, we begin by putting every type description in a standard form that takes care of "obviously unimportant" distinctions like those above

Copyright © 2005 Elsevier

Type Checking

- Structural equivalence depends on simple comparison of type descriptions substitute out all names
  - expand all the way to built-in types
- Original types are equivalent if the expanded type descriptions are the same

Copyright © 2005 Elsevier