The Ada Programming Language

Michael Beaver

**History**

* Result of a design competition started by the U.S. Department of Defense (DoD) in 1975
* Requirements documents: Strawman, Woodenman, Tinman, Ironman, and Steelman
* Winning proposal design by an international team led by Jean Ichbiah (from France)
* Pascal-based design
* First standardized in 1983
* Became *the* language for DoD software projects (for a time, at least)
* Named for Ada, Countless of Lovelace

**Global Issues**

* International team—the process from 1975 actually involved Great Britain and France
* Used all over the world in different embedded and life-critical systems
  + Australia to Kenya to Vietnam—lots of people use Ada
* Support for non-English characters (i.e., Unicode)

**Design Goals**

* Reliability
* Maintainability
* Programming as a human activity
  + Emphasis on readability over writeability
  + Programming should be sensible and appeal to users’ intuition
* Programs should be efficient, no matter the cost
* Since 1995, the language has focused on becoming more flexible and extensible

**Syntax**

* Highly readable, though not as writeable
* Reserved words (over 70)
* Essentially supports the standard ASCII character set and Unicode (see above)
  + Note: Ada does not use square brackets or curly braces
* Programs are formed from compilation units, which consist of lexical elements
  + Lexical elements are delimiters, identifiers, reserved words, literals, or comments
* Comments are denoted by double hyphen and extend to the end of the line
  + No support for block comments
* Identifiers must be unique
* Ada is case *insensitive*
* Delimiters are simple or compound
  + Simple example: space character (separator)
  + Compound example: := (assignment)
* Support for language-defined and user-defined pragmas

**Data Types and Type-Checking**

* Elementary or composite (e.g., a record)
* Variable and constant data require declarations
* Objects are data containers created at runtime that have an associated type and value
* Scalar types
  + Discrete
    - Integers
    - Enumerations
  + Numeric
    - Integers
    - Reals (floating and fixed)
* Support for abstract types
* Support for pointers (access types)
* *Strongly* typed—all checking is done at compile-time
* Data types have built-in operations: attributes (e.g., Integer’Image(MyInt))

**Data Structures**

* Arrays
  + Linear representation with upper and lower bounds
  + One-dimensional or multi-dimensional
  + Accessed using parentheses with integer (or enumerated) index
  + May be sliced to create another array
* Records
  + Composed of named components (but may be null)
  + May have variant parts
    - Changes the record based on some condition (see: case statements)
  + May have discriminants
    - Parameterize the type of the record
* Other data structures (e.g., stack) can be made using packages

**Scoping**

* Very specific scoping rules
* Identified entities have scope only within the regions in which they are declared
  + Regions: blocks, loops, exception handlers, and so on
  + Declarations are local to their declarative region
  + Declarations are global to encapsulated regions
  + Basically, an object has scope within its enclosing region
* Declarations may be overridden
  + Very restricted usability
  + One declaration can be overridden by another with the same name
* A declaration is immediately visible in its immediate scope
* A declaration is use-visible if it is made visible by a use clause
* Entities may be renamed in declarations by using the renames reserved word

**Names and Expressions**

* All declared entities, subprograms, and objects have names (essentially an identifier)
  + Expanded names can be used to select components of types, entries to tasks, etc.
  + Names must be unique within scope
  + Names must start with an alphabetic character or underscore and may contain any number of characters, including numeric digits
* Expressions are essentially composed of identifiers (names), delimiters, operators, and subprogram calls
  + Support for literals: characters, strings, and numeric
  + Support for component aggrandizement (e.g., with records’ components)
  + Expressions are either arithmetic or conditional
  + Many built-in operators, including equality (=) and inequality (/=)

**Statements and Control Structures**

* Statements define actions to be performed during execution
  + Simple or compound
    - Simple: do *not* enclose other statements
    - Compound: do enclose other statements
  + Statements themselves may have identifiers (e.g., for a loop)
* Assignment (:=) *only* stores a value in an object (of compatible type)
* Conditional statements
  + if, elsif, and else
  + case
* Loops
  + generic loop
  + for loop
  + while loop
  + May be prematurely exited using exit statement
* Blocks
  + Enclosed by begin and end
  + Fundamental to the language (also inherited from Pasal!)
* goto statement support
  + Transfers control to a label enclosed in double angle brackets (e.g., <<Here>>)

**Subprograms and Parameter Passing**

* Procedures
  + Do *not* return a value
* Functions
  + *Must* return a value
* Subprograms have two parts: specification and body
* Subprograms may enclose other subprograms (i.e., nested subprograms)
* Three parameter passing modes:
  + in : read in, default (implied)
  + out : write out
  + in out : read in and write out (similar to C++ pass-by-reference)
* Parameters may have default values
* Return statements may be simple or extended
  + Extended return statements enclose other statements to execute on return
* Operators may be overloaded—operators are simply function calls
* Special subprogram types:
  + Null procedure
  + Expression function

**Packages**

* Group logically related program entities
* Typically contain declaration of a type and primitive subprograms of that type
* Packages are new—not found in Pascal
* Packages have two parts: specification and body
  + Unlike with subprograms, package bodies are optional
  + Package specifications are publicly visible
  + Package bodies are private
* Ada’s answer to object-oriented design and programming
  + Similar to C++’s class header and implementation constructs

**Generic Units**

* Generic packages or subprograms
* May be parameterized and instantiated in other program units
  + Must be instantiated before using
* Generic units have two parts: declaration and body
  + Declaration denotes whether a package or a subprogram (and formal parameters)
  + Body contains the actual package body or subprogram body
* Formal packages and subprograms may be passed as parameters to generic units!
* Similar to templates in C++

**Program Structure**

* An Ada program consists of a set of partitions, which consist of compilation units
  + These units may execute concurrently and in different address spaces
  + *Separate* compilation of program units
  + Library units: packages, subprograms, etc. with children or nested units
    - Included using use and with clauses
* Separate compilation is supported through subunits, which identify parent units
* Compilation units are compiled within certain contextual environments, which help govern the visibility (i.e., the scope) of the units

**Exceptions and Exception Handling**

* Exceptional exception handling facilities!
* Impressive collection of built-in, language-defined exceptions
* Exceptions may be raised explicitly or may arise naturally
* Exceptions are handled by an exception handler at the end of a block
  + Exceptions may also be propagated to the enclosing block

**Concurrency**

* Concurrency is achieved with tasks
  + Tasks represent separate, concurrent threads of control
  + Tasks may communicate synchronously or asynchronously
  + Tasks have two parts: declaration and body
  + Tasks proceed in stages (inactive, active, terminated, etc.)
* All tasks, except the overall environment task, are dependent on a master task
  + Master tasks are blocked until slave tasks have complete and terminated
* Protected objects may be used to access shared data (via protected operations)
* Tasks communicate primarily via calls on entries and via protected subprograms
  + Calls on entries offer blocking of the caller until a condition is met
  + Protected subprograms offer access to shared data objects
* Operations on tasks are queued via entry and accept statements
  + A “rendezvous” occurs when a called task accepts the calling task’s message
* Tasks may be terminated abnormally by using the abort statement

**Recursion**

* *All* subprograms may be recursively called
* Packages and types may be recursively defined, per se
  + This necessitates an incomplete forward declaration, a pointer, and then a complete declaration using that pointer
  + Not the “cleanest” method
* Recursion may be disabled via the Restrictions(No\_Recursion) pragma

**Input and Output**

* Support for:
  + Text I/O
  + File I/O
  + Memory buffer I/O
  + Streams
* Separate package just for I/O exceptions

**Unusual Features**

* Type attributes
* Extended return statements
* Ada can interface with C, COBOL, and FORTRAN

**Contributions to the Programming Language Landscape**

* Packages
* Very robust exception handling
* Task concurrency

**Promise for the Future**

* The 2012 standard is the newest, but the designers are musing new features for the next standard, including shorthand assignment statements like those found in C++ (e.g., +=)