V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 9

—Slide 1—

The ADA Programming Language
Language Survey 3

- *Countess Ada Augusta Lovelace*
  First programmer (hacker) on Babbage's *Analytical Engine*.
- Relatively new programming Language (1980) developed by DoD. Currently, the only language approved for DoD software project
- Developed to reduce high cost of *designing*, *writing* and *maintaining* DoD software.
- Major items of software are for "**embedded systems**"
  - Primary purpose is control.
  - Incorporated in larger systems.
  - Large software: 50,000–100,000 lines of code
  - Long-lived systems (10–15 years)
  - Continuously evolving.

—Slide 2—

The Design History

- 1975: HOLWG (High Order Language Working Group) was established to investigate problems of developing common language for DoD.
- 1975–1978: List of requirements: *STRAWMAN*, *WOODENMAN*, *TINMAN*, *IRONMAN*, *Revised IRONMAN*, and finally *STEELMAN*
- Design Competition:
  Winner: Jean Ichbiah, Cii Honeywell Bull
- 1987: Ada ISO Standard 8652
- 1983–1987: Analysis of Technical Queries by ARG (Ada Rapporteur Group). Resulted in AIs (Ada Issues).
- 1988: DOD (and ANSI) established Ada 9X project.
- 15 Feb 1995: Ada 95. Core Language + Specialized Annexes

—Slide 3—

Quick Overview (Ada 83)

- Strong Typing
  Scalar, Composite, Access, Private & Derived
- Representation Specification
- Standard Control Constructs:
  Structured Language–Pascalish
- Subprograms: Procedures & Functions
- Program Structuring Facilities: Packages
- Generics: Polymorphisms Extends the concept of type and functional abstractions
- Exceptions
- Separate Compilation
- Tasking (Rendezvous)

—Slide 4—

Quick Overview (Ada 95)

- Object Oriented Programming
- Program Libraries

  (Replaces Ada 83's flat structure by a hierarchy—Visibility Control and Program Extension without recompilation)
- Interfacing

  Graphical User Interface (GUI)—Program Call-back.
- Tasking (Static Monitor)

—Slide 5—

Ada Type System

- Programmer Defined Types
  Set of values, Applicable set of operations and set of properties, accessible via `attributes`
- **Type Definition**

```
type <identifier> is <type-definition>
```

- **Type Binding**

```
<variable> : <type>
```

- **Attributes**

```
<name>'<attribute-identifier>
```

—Slide 6—

Examples

```
type DAY is (MON,TUE,WED,THU,FRI,SAT,SUN);
   --Enumeration Type
TODAY: DAY; TOMORROW: DAY; CURRENT_DAY: DAY;

HOURS_WORKED: array(DAY'FIRST..DAY'LAST)
                   of INTEGER;

if TODAY <= FRI then
   TOMORROW := DAY'SUCC(TODAY);

for CURRENT_DAY in DAY'FIRST..DAY'LAST loop
   ...
end loop;
```

—Slide 7—

Primitive Scalar Types

- Discrete Types
  i) Enumeration Types, ii) Character type,
  iii) Boolean Types, iv) Integer Types
- Real Types
  i) Fixed-Point Types, ii) Floating-Point Types
- **Note:** Integer and Real types together form the *numeric types*.
- `INTEGER` = Predefined integer types
  `SYSTEM.MIN_INT..SYSTEM.MAX_INT`
- Operation
  `+`, `-`, `*`, `/` (Arithmetic operations)
  `<`, `<=`, `=`, `/=`, `>=`, `>` (Logical operations)

—Slide 8—

Discrete Types

- **Note:** Characters & Boolean are predefined *enumeration types*

  Characters = 128 ASCII characters: 'A', 'B', 'C', ...

  Booleans = `FALSE` & `TRUE`

- All discrete types are **ordered**: `FALSE` < `TRUE`.

- Discrete type values can be used for **indexing** & **iteration over loops**.

- Attributes of Discrete Types (e.g., `T`)

  `T'POS`: Position Number

  ```
  SUIT'POS(HEARTS) = 0
  ```

  `T'VAL`: Inverse of `POS`

  ```
  SUIT'VAL(0) = HEARTS
  ```

  `T'SUCC`, `T'PRED`

  `T'FIRST`, `T'LAST`

—Slide 9—

Real Type

- **Floating Point:**
  Accuracy is fixed by a *relative* error bound

  ```
  type WEIGHT is digits 10;
  ```

  Values have an accuracy of 10 digits.

- **Fixed Point:**
  Accuracy is fixed by a *relative* error bound

  ```
  type VOLTAGE is delta 0.1 range 0.0..1.0;
  ```

  Values have an accuracy at least as fine as 0.1.

- Some Attributes
  `FX` = Fixed Point: $(-2^N + 1)\delta..(2^N + 1)\delta$

  ```
  FX'DELTA, FX'LARGE
  ```

  `FL` = Floating Point: sgn $*$ mantissa $* 2^E$; mantissa has B digits.

  ```
  FL'DIGITS, FL'MANTISSA, FL'EMAX,
  FL'SMALL, FL'LARGE, FL'EPSILON
  ```

—Slide 10—

Derived Type

- Specific Type + *Constraints*
  E.g., Range Constraints for scalar
- Examples

```
subtype BYTE_SIZE is INTEGER range -2**7..2**7;
subtype CAPS is CHARACTER range 'A'..'Z';
subtype <identifier> is <parent-type> range <constraint>;
```

- **Object Declaration**

```
X, Y: constant INTEGER range 0..128 = abs(N);
<identifier>: <mutability> <type> <constraint>
        = <init-value>;
```

- Range constraint can be tested at runtime

```
if CURRENT_INPUT not in CAPS then ...
```

—Slide 11—

Assignment Statement

```
X := Y;
```

- **Types must match**
  - Same type name–Not structure. Ada uses *name equivalence*.
  - Type checking is at compile time.
  - Type mismatch ⇒
    Program considered *illegal*.
- **Subtype constraints**
  - Compatibility condition: type(X) $\geq$ type(Y)
  - Constraint checking is at run time.
  - Violation raises `constraint` *exception*.
    (Program is **not** illegal.)

—Slide 12—

Examples

```
subtype NATURAL is INTEGER range 1..INTEGER'LAST;
A: INTEGER;
B: FLOAT;
C: NATURAL;
D: INTEGER range 0..INTEGER'LAST;

A := B;                --illegal
A := INTEGER(B);       --type conversion, legal
A := C;                --legal
A := D - 3;            --legal
A := C + INTEGER(B);   --legal
C := D;                --constraint exception
C := A;                --constraint exception
```

[End of Lecture #9]