V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 10

—Slide 1—

The ADA Programming Language
Arrays

- **Arrays in Ada:**
  - Fixed Size—Type may be *unconstrained* at definition But bounds must be given at object declaration.
  - Elements are all of same *subtype*
  - Permits: Assignment, Equality Testing, Slicing, . . .

```
subtype NATURAL is INTEGER range 1..INTEGER'LAST;
type SHORT_STRING is array(1..10) of CHARACTER;

type STRING is
     array (NATURAL range <>) of CHARACTER;

NAME: STRING(1..10);
```

—Slide 2—

Array Assignment

- Assigning array `B` to `A`: `A := B`
  1. Legal, if type of `A` is same as type of `B`.
  2. If `A` has same number of elements as `B`, then `B` is copied into `A` *positionally*—Otherwise, `constraint-error exception` is raised.

```
declare
  A: STRING(1..10); B: STRING(11..20);
begin
  A := B;
end;
```

- **Array Attributes**

`A'LENGTH`	Number of elements
`A'FIRST`	Lower Bound
`A'LAST`	Upper Bound
`A'RANGE`	subtype `A'FIRST..A'LAST`

—Slide 3—

Array Indexing & Slicing

- Array Indexing:

```
S: STRING(1..10);
S(3) := S(2);
```

- Array Slicing (1D arrays only)

```
S(3..7)               --an array object
S(3..7) := S(4..8);
S := S(2..10) & S(1) -- & == concatenation opn
```

- Array Aggregates:

```
type SYM_TAB is array (CHARACTER range <>) of INTEGER;
TABLE: SYM_TAB('a'..'c') := (0, 1, 2);
TABLE := ('c' => 2, 'b' => 1, 'a' => 0);
TABLE := ('c' | 'b' => 1, others => 0);
```

—Slide 4—

Records

- **Records in Ada:**
  - *Heterogeneous*: Components need not be of same type.
  - Fields are accessed by component names: E.g., `MY_CAR.CAR_MAKE`
  - *Variant Records* Tag (discriminant) fields cannot be changed at run-time.
  - Permits: Assignment and Equality Testing.

```
type CAR_MAKE is (FORD, GM, HONDA);
subtype CAR_YEAR is INTEGER range 1900..1999;
type CAR is
  MAKE: CAR_MAKE;
  YEAR: CAR_YEAR;
end record;

MY_CAR: CAR;
```

—Slide 5—

Records (Contd)

- Records may be *nested*...*initialized* at declaration.
- A record `B` may be assigned to record `A`, provided they have same type.

```
A, B: CAR;
A := B;
```

- Record Aggregates:

```
YOUR_CAR: CAR :=
   (GM, 1981);
```

```
YOUR_CAR: CAR :=
   (MAKE => GM,
    YEAR => 1981);
```

—Slide 6—

Variant Records

- Similar to PASCAL variant records:
- Except—Type declaration only defines a template; When object is declared, *discriminant value must be supplied*.

```
type VEHICLE_TAG is (CAR, TRUCK);
type VEHICLE(TAG: VEHICLE_TAG) is record
  YEAR: MODEL_YEAR := 93;
  case TAG is
    when CAR => COLORS: COLOR_SCHEME;
    when TRUCK => AXLES: NATURAL;
  end case;
end record;

YOUR_TRUCK: VEHICLE(TRUCK);
REFRIGERATOR: VEHICLE;      --Illegal
```

- There may be more than one discriminant...But they must all be of discrete types....Discriminant can be used as an uninitialized constraint.

```
type BUFFER(LENGTH: NATURAL) is record
  POOL: STRING(1..LENGTH);
end record;
```

—Slide 7—

Access Types

- Allow manipulation of pointers.
- Allow control of object creation.

```
type STRING_PTR is access STRING;
type STRING_10_PTR is access STRING(1..10);
P, Q: STRING_PTR; P10: STRING_10_PTR;

P10 := new STRING(1..10);
P10 := new STRING(2..11);     --Constraint Error
P10 := new STRING;            --Illegal

P := new STRING(1..3);        --OK
P.all := "BUD";
Q := new STRING("MUD");
P := Q;
P.all := Q.all
```

- `new` creates a new object that can be designated by the access type.

—Slide 8—

Recursive Types

```
type NODE;  --Incomplete Declaration;
type NODE_PTR is access NODE;

type NODE is
  record
    DATUM: CHARACTER;
    NEXT: NODE_PTR;
  end record;
```

—Slide 9—

Generalized Access Types

- Inherent Access to declared objects
  (Not just objects created by allocators)

```
type INT_PTR is access all INTEGER;

IP: INT_PTR;
I: aliased INTEGER;
IP := I'Access
```

- **Note:** Designated variable must be marked `aliased`.
- `Access` attribute is only applicable to an object whose lifetime is at least that of the access type.
- Avoids "dangling reference" problem.

```
type CONST_INT_PTR is access constant INTEGER;

CIP: CONST_INT_PTR;
C: aliased constant INTEGER := 1815;
CIP := C'Access
```

- Access is restricted to read-only

—Slide 10—

Control Structures

- Assignment Statements

```
DISCRIM := (B**2 - 4.0*A*C);
TABLE(J) := TABLE(J) + 1;
VECTOR := (1..10 => 0);
```

- Conditional Statements

```
if (A=1) then
  ...
end if;

if (A=1) then
  --...
elsif (A=2) then
  --...
else
  --...
end if;
```

```
case A is
  when 1 => --...;
  when 2 => --...;
  when others => null;
end case;
```

—Slide 11—

Control Structures: Iteration Clause

- Iteration Statements—Basic Loop

```
loop
  -- Statements to be repeated
end loop;
```

- *Iteration Clause*
  Execution of a basic `loop` terminates when
  1. *The iteration is completed* or
  2. *A* `loop exit` *statement is executed*

```
SUM := 0;                  SUM := 0;
for I in 1..10 loop        for I in reverse 1..10 loop
 SUM := SUM + A(I);          SUM := SUM + A(I);
end loop;                  end loop;

SUM := 0; I := 1;          SUM := 0; I := 1;
while I <= 10 loop         loop
  SUM := SUM + A(I);         exit when I > 10;
  I := I + 1;                SUM := SUM + A(I);
end loop;                    I := I + 1;
                           end loop;
```

—Last Slide—

A Complete Ada Program

```
with I_O_PACKAGE;
procedure TEMPERATURE_CONVERSION is
  use I_O_PACKAGE;
  -- Convert temp in Fahrenheit to Celsius

  FAHRENHEIT_TEMP; CELSIUS_TEMP: FLOAT;
begin
  GET(FAHRENHEIT_TEMP);
  CELSIUS_TEMP := (FAHRENHEIT_TEMP - 32.0)*5.0/9.0;
  PUT(CELSIUS_TEMP);
end TEMPERATURE_CONVERSION;
```

[End of Lecture #10]