Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: FK pp. 115-151

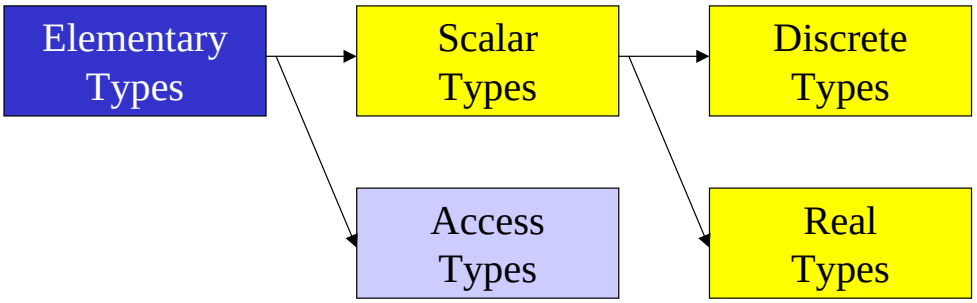
Lecture 8
Sept 17 2003

Types

- Type
  - A set of **values**
  - A set of **primitive operations**
- Grouped into classes based on the similarity of values and primitive operations
  - **Elementary** types
  - **Composite** Types

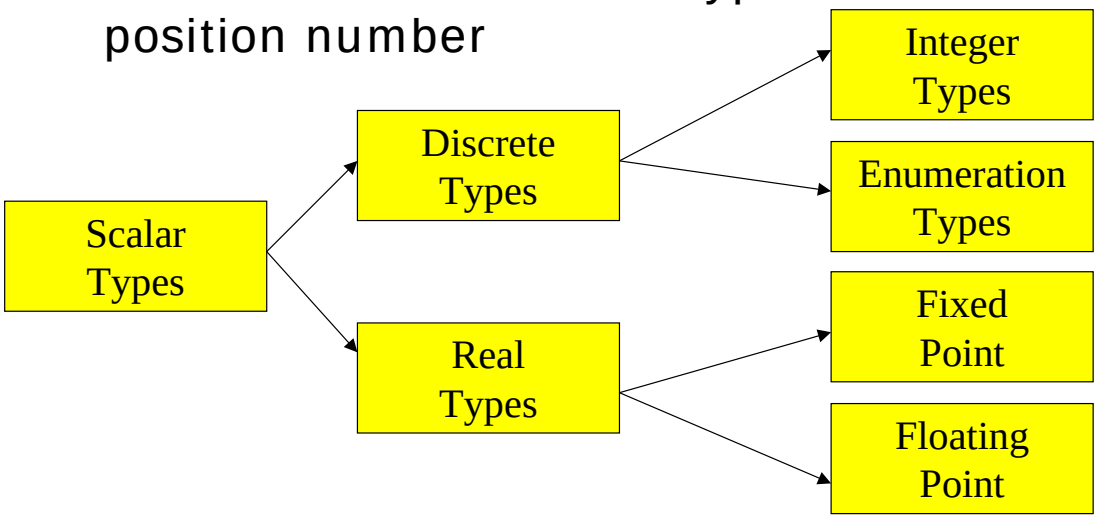
Type Classification

- **Elementary** Types : Values are logically indivisible
- **Composite** Types : Values composed from components



Scalar Types

- Ordered → relational operators are defined
- Each value of a discrete type has a position number



Attributes of Scalar Types

- S'First denotes the lower bound of the range of S. The value of this attribute is of the type of S.
- S'Last denotes the upper bound of the range of S
- S'Range is equivalent to the range S'First .. S'Last

Operations on Scalar Types

- S'Min returns lower of two elements
- S'Max returns higher of two elements
- S'Value accepts a string and returns the value in the type
- S'Image converts the value into a string
- S'Pred and S'Succ – behavior depends on the scalar type
  - S'Pred (Integer) : returns (Integer -1)
  - S'Succ (Integer) : returns (Integer + 1)

Subtypes

- A **subtype** is a subrange of a larger type.
- Subtypes of the same larger type are *not* distinct types. A subtype and the larger type are also not distinct types. Thus subtypes of the same thing are assignment-compatible.
- The benefit of subtypes is that range checks avoid some nonsense.

Subtype Example

- Two useful sub-types of the integers are built into Ada:
  -
```
subtype POSITIVE is INTEGER range 1..INTEGER'LAST;
subtype NATURAL is INTEGER range 0..INTEGER'LAST;
```
- Subtypes are appropriate whenever there are ranges of allowed values.
  -
```
min_on_bus : constant := 0;
max_on_bus : constant := 80;
type no_on_buses is range min_on_bus .. max_on_bus;

max_seated : constant no_on_buses := 50;

subtype seated_on_buses is no_on_buses
          range min_on_bus .. max_seated;
subtype standing_on_buses is
          range min_on_bus .. (max_on_bus - max_seated);
```

Subtypes

```
subtype Natural is Integer range 0..Integer'Last;

subtype Positive is Integer range 1..Integer'Last;

subtype NonNegativeFloat is Float range 0.0 .. Float'Last;


subtype SmallInt is Integer range -50..50;

subtype CapitalLetter is Character range 'A'..'Z';
X, Y, Z      : SmallInt;
NextChar     : CapitalLetter;
Hours_Worked : NonNegFloat;


X := 25;
Y := 26;
Z := X + Y;
```

Operations on Discrete Types

- S'Pos(Arg) returns the position number of the argument
- S'Val(Arg) a value of the type of S whose position number equals the value of S

CQ

- The outputs are exactly the same
- There will be no outputs
- The outputs are different
- I don’t know

Enumeration Types

- A data type whose values are a collection of allowed words

```
type Class is
  (Freshman, Sophomore, Junior, Senior);

type days is (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
type colours is (white, red, yellow, green, blue, pink, black);
type traffic_colours is (green, yellow, red);
type suits is (clubs, diamonds, hearts, spades);
```

Enumeration Types

- Enumeration types have the following benefits:
  - readable programs
  - avoid arbitrary mapping to numbers
    - e.g. better to use "Wed" than 3 for a day of the week
  - they work well as selectors in case statements
- Example: mix_colours.adb

Attributes of Enumerated Types

```
type Days is
  (Monday, Tuesday, Wednesday, Thursday, Friday,
   Saturday, Sunday);

Today    : Days; --current day of the week
Tomorrow : Days; --day after Today


Today := Friday;
Tomorrow := Saturday;


Days'First           is Monday
Days'Last            is Sunday
Days'Pos(Monday)     is 0
Days'Val(0)          is Monday
Days'Pred(Wednesday) is Tuesday
Days'Pred(Today)     is Thursday
Days'Succ(Tuesday)   is Wednesday
Days'Succ(Today)     is Saturday
```

You must ensure the result is legal. A CONSTRAINT_ERROR will occur at run-time otherwise. For example, `days'SUCC(Sun)` is illegal.

Derived Types

- age := -20;
- height := age - class_size;
- shoe_size := 2 * no_on_bus;

- Types help program values reflect the real world.

Derived Integer Types

- New data types can be **derived** from INTEGER:
-
```
type ages is new INTEGER range 0 .. 110;
age : ages;
voting_age : constant ages := 18;

type heights is range 0 .. 230;
height : heights;

min_enrolment : constant := 6;
max_enrolment : constant := 200;
type class_sizes is range 0..max_enrolment;

class_size : class_sizes;
```

Type conversion

- Ada has *strong typing:* different types cannot be mixed
- Explicit type conversion is permitted:

- ```
  type length is digits 5 range 0.0 .. 1.0E10;
  type area is digits 5 range 0.0 .. 1.0E20;

  function area_rectangle (L,H : length) return area is
  begin
     return area(L) * area(H);
  end;
  ```

Benefits of derived types

- Nonsense rejected by compiler
  - `height := age - class_size;`
- "Out of range" rejected by compiler
  - `age := -20;`
- “Out of range” run time error
  - `class_size := class_size + 100;`
- Enforce distinct nature of different objects
- Robust, elegant, effective programs
  ```

I/O Libraries

- Each distinct type needs its own I/O library.
- General form:
  - **package** type_io **is new** TEXT_IO.basetype_io (typename);

```
package int_io is new TEXT_IO.INTEGER_IO (INTEGER);

type ages is new INTEGER range 0 .. 110;
package ages_io is new TEXT_IO.INTEGER_IO (ages);

type measurement is digits 10;
package measurement_io is new TEXT_IO.FLOAT_IO (measurement);

type suits is (clubs, diamonds, hearts, spades);
package suits_io is new TEXT_IO.ENUMERATION_IO (suits);

type colours is (white, red, yellow, green, brown, blue, pink, black);
package colours_io is new TEXT_IO.ENUMERATION_IO (colours);
```

Input/Output Operations

```
type Days is
  (Monday, Tuesday, Wednesday, Thursday, Friday,
   Saturday, Sunday);

package Day_IO is new Ada.Text_IO.Enumeration_IO(Enum=>Days);

if this_day in weekend_days then
   put(”Holliday!”);
end if;

Day_IO.Get(Item => Today);
Day_IO.Put(Item => Today, Width => 10);
```

Example

- subtypes[1..3].adb