Lecture 7: References and Assignments

Object structure

An object is made of fields
Each field is a value, which is either:

A basic value: integer, character, "real" number... (known as an expanded value)

A reference to another object

Two kinds of type

Reference types; value of any entity is a reference.
Example:

Expanded types; value of an entity is an object.
Example:

Classes and objects

At run time: objects (software machines)

In the program text: classes
Each class describes a set of possible run-time objects.

Expanded classes

A class may be declared as

expanded class E_STATION
... The rest as in STATION...

Then any entity declared

d : E_STATION

has the expanded semantics just described.
Basic types as expanded classes

expanded class INTEGER...
expanded class BOOLEAN...
expanded class CHARACTER...
expanded class REAL...
expanded class DOUBLE...

\[ n : \text{INTEGER} \]

Strings are objects

The name field is a reference field

Initialization

Automatic initialization rules:
- 0 for numbers (integers, reals)
- "Null" character for characters
- False for booleans
- Void for references

These rules apply to:
> Fields (from class attributes), on object creation
> Local variables, on start of routine execution (includes Result)

Fields reflect attributes of the class

An attribute

Another attribute

References may cause cycles

Setting fields (in routines of the class)
Setting fields (in routines of the class)

```pascal
class POSITION
feature - Access
  x: REAL -- Horizontal position
  y: REAL -- Vertical position
feature - Element change
  set(xval, yval: REAL)
    require x_positive: xval >= 0
    require y_positive: yval >= 0
    do
      x := xval
      y := yval
    ensure
      x_set: x = xval
      y_set: y = yval
  endend
```

What you may do

```pascal
class METRO_STATION
feature
  x, y: REAL -- Coordinates of center of metro station
  size: REAL -- Size of bounding square
  upper_left: POSITION -- Upper-left position of bounding square
feature
  adjust_positions
    do
      upper_left.set (x – size/2, y + size/2)
    end
end
```

Feature calls

In POSITION, we assume:

```pascal
set(xval, yval: REAL)
  do
  end
```

In another class, e.g. METRO_STATION:

```pascal
upper_left.POSITION
  adjut_positions
    do
      upper_left.set (x – size/2, y + size/2)
    end
end
```

The system execution process

The current object

At every moment during execution, operations are being performed on a current object.

Initially: the root object. Then:

- An unqualified call such as `set(u, v)` applies to the current object.
- A qualified call such as `x.set(u, v)` causes the object attached to `x` to become the current object. After the call the previous current object becomes current again.

To denote the current object: use `Current`.
Current object

At any time during execution, there is a current object, on which the current feature is being executed.

Initially it is the root object.

During a "qualified" call $x.f(a)$, the new current object is the one attached to $x$.

At the end of such a call, the previous current object resumes its role.

"General relativity"

Feature calls

In `METRO_STATION`, `adjust_positions` is:

```plaintext
adjust_positions is
  do
    upper_left.set(x - size/2, y + size/2)
  end
upper_left: POSITION
```

Entities

An entity is a name in the program that denotes possible run-time values.

Some entities are constant.

Others are variable:
- Attributes
- Local variables

The client relation

Because class `METRO_STATION` has a feature

```plaintext
upper_left: POSITION
```

(and calls of the form `upper_left.set(...)`) the class `METRO_STATION` is a client of class `POSITION`.

Changing variable values: assignment

Target is an expression:
- Call to a query:
  - `position`
  - `upper_left.position`
- Arithmetic or boolean expression:
  - $a + (b * c)$
  - $(a < b)$ and $(c = d)$

Target may be:
- An attribute
- Result in a function
- A "local variable" of a routine (not yet seen)
Assignment

Replaces a value by another

```
p
x  0
y  0
```

```
p.set(2, 1)
```

Setting fields (in routines of the class)

class POSITION
feature - Access
  x: REAL -- Horizontal position
  y: REAL -- Vertical position
feature - Element change
  set(xval, yval: REAL) is
    do
      x := xval
      y := yval
    end
end

Assignment

class METRO_STATION
feature
  location: POSITION
  name: STRING
  length: REAL
set_all(p: POSITION; l: REAL; n: STRING) is
  do
    location := p
    length := l
    name := n
  end
end

Do not confuse assignment with equality

```
x := y
if x = y then...
if x = Current then...
```

A linked list of strings: inserting at the end

```
Haldenegg -> Central -> Hauptbahnhof
```

Effect of an assignment

Reference types: reference assignment
Expanded types: value copy

```
class TWO_VALUES
  feature
    item: INTEGER
    right: TWO_VALUES
set(n: INTEGER; r: TWO_VALUES) is
  -- Reset both fields
  do
    item := n
    right := r
  end
end
```

```
t: TWO_VALUES
create t
```

```
t.set(25, Void)
```

```
item := n
right := r
```
Inserting an item at the end

```
extend(v: STRING) is
    -- Add v to end.
    -- Do not move cursor.
    local
        p: LINKABLE [STRING]
    do
        create p.make(v)
        if is_empty then
            first_element := p
            active := p
        else
            last_element.put_right(p)
            if after then active := p end
        end
        last_element := p
        count := count + 1
    end
```

**LINKABLE** cells

```
class LINKABLE feature
    item: STRING
        -- Value in this cell
    right: LINKABLE
        -- Cell, if any, to which this one is chained
    put_right(other: like Current) is
        -- Put other to the right of current cell.
        do
            right := other
        ensure
            chained: right = other
        end
end
```

Local variables (in routines)

A form of entity (they are also called "local entities")
Just declare them on entry to a routine:

```
r(...) is
    -- Header comment
    require...
    local
        x: REAL
        m: METRO_STATION
    do
        ... Can use x and m here ...
    ensure...
end
```

Local variables include Result for a function

Reading assignment

Chapter 6 (control structures), up to 6.6.

Exercise (uses loops)

Reverse a list!

```
(linked list)

3 count

last_element

first_element

Haldenegg item right

Central Hauptbahnhof

Paradeplatz
```

What we have seen

Inserting an item at the end

**LINKABLE** cells

Local variables (in routines)
What we have seen

- The current object
- Expanded vs reference types
- Assignment:
  - For references
  - For expanded values
- Linked data structures
- A glimpse of conditional instructions

End of lecture 7