4A Data-Items and Data Types

I have preliminary put this page around Chapter 4, since I think it is needed here, but it is up for discussion. The same for the Chapter on 4B Statements – see also the note there!

```
I guess that the structure of this chapter should be like this:

Data items and their types   -- ikke 'and Value Types', da referencer er her
   Reference data items
   Value data items
      Primitive values types
      Composite value types

OLM: Men skal man starte med references?
Og hvor tænker du der er et afsnit om Values and Objects?
```

In this Chapter, we summarize [redacted] data-items that may represent primitive values, string values and references to objects.

As mentioned in previous chapters, objects may have attributes in the form of data-items that may represent properties characterizing the phenomena. For the `Account` example, we have seen examples of data-items such as `owner`, `balance`, and `interestRate`.

There is a fundamental difference between the data-items `owner` and `balance` of `Account` objects. While `owner` is a *reference* to the object representing the owner of the account, `balance` represents a property that is given by a *value*.

Also, some value types are denoted *primitive values*. `balance` and `interestRate` are examples of data-items holding primitive values, in this case of type `Real`.

References

[redacted]The data-item `owner` is holding values of type `String`, but is usually not considered a primitive value type. The reason is that the primitive values may be represented in a fixed number of bytes in the computer whereas a `String` value may need a variable number of bytes for its representation.

As mentioned, data-items like Account_1010, aClerk, and aCustomer are examples of data-items holding references to objects. These data-items were declared as:

```
Account_1010: obj
   ...
aClerk: obj
   ...
aCustomer: ref Customer
```

The data-items `Account_1010`, and `aClerk`, are constant in the sense that they refer to the same object during the lifetime of the program execution. The data-time `aCustomer` is variable since it may refer to different `Customer` objects during the program execution.

```
JohnSmith: obj Customer("John Smith")
aCustomer: ref Customer
aCustomer := Customer("Mary Poe")
...
aCustomer := JohnSmith
```

In the above example, `JohnSmith` is constantly referring to the object `Customer("John Smith")` whereas `aCustomer` may refer different `Customer` objects. It is first assigned a new object `Customer("Mary Poe")`. Then later it may be assigned `JohnSmith`, which nimplies that `Mary` refers to the same object as `JohnSmith`.

This page should be split into subpages.

Value data items

+++Value data items: name and value type

Primitive value types

Data-items such as `owner`, `balance` and `interestRate` are examples of data-items holding values of *primitive value types*. These data-items are declared as follows:

```
owner: var String
balance: var Real
interestRate: var Real
```

The specification `var String` in the declaration of `owner`, specifies that `owner` may hold string values, and `String` is called the *type* of `owner`.

The specification `var Real` in the declarations of `balance` and `interestRate` specifies that these may hold real values, and their type is thus `Real`.

`Real` and `String` are examples *value types*, which are similar to classes. We discuss value types in Chapter *Objects and Values*.

In the following sections we describe other common primitive values.

Real values

The data-items `balance` and `interestRate` are as mentioned examples of data-items that represent real values. Examples of real values are: 100, 3.14, 0.56, -12.11, `13.5E7`, and `1.11E-7`. The `'E'` in `13.5E7` means that the value is 13.7 multiplied by 10^7.

The keyword **`var`** used in the declaration specifies that these data-items are variable, which means that they may be assigned different values during the execution of the program:

```
balance := 500
interestRate := 3.07
...
balance := balance + amount
interestRate := 2.9
...
```

It is also possible to specify that a data-item holds a value which is constant during the whole of the program execution:

```
X: val 3.1
```

The keyword **`val`** specifies that the data-item `X` is constant and it holds the value `3.14` during the whole of the program execution.

The set of real numbers is infinite, and a computer can only represent a subset of the real values. This subset is often called *floating point values*. Since only a subset can be represented, the evaluation of floating point expressions often imply that some values are rounded to a value that the computer can represent. For calculations of a large amount of numbers these rounding may lead to misleading and erroneous results. It is therefore recommended that people writing programs involving floating point calculations learn about how to try to avoid rounding errors.

The above paragraph may be placed in a side box

Integer values

We need an example of a integer data-item in one of the previous chapters.

In the example in Chapter/Section xx.yy, `anInt` is an example of a data-item that may represent an `integer` value. Integer is another example of a value type representing primitive values. Examples of integer values are 17, 111, -4, 0, and -1014.

An for Real, an integer may redeclared as a variable or a constant:

```
anInt: var Integer
aConst: val 117
```

The data-item `anInt` is an example of a variable that may hold different integers during the program execution. the data-item `aConst` on the other hand is constant and holding the value 117 during the whole programm execution.

Boolean values

We need an example using booleans.

A data-item may be of type `Boolean` and such a data-item may have the value `True` or `False`. A data-item of type `Boolean` may be declared as follows:

```
B1: val True
B2: var Boolean
```

The data-item `B1` is a constant having the value True during the whole program execution. The data-item `B2` is a variable that may hold the values True or False at different point during the program execution. In the next example, `B2` is assigned the value False

```
B2 := False
```

Character values

We need an example using char

A data-item of type Char may hold a value representing a character from the UniCode character set. A character may be a letter, a digit, a special character, a blank/space or an end-of-line character. A character literal is typically written within single quotes like:

```
‘a’, ‘b’, ‘Q’, 'L’  – examples of letters
‘0’, 7′             – examples of digits
‘.’, ‘:’, ‘+’,      – examples of special characters
‘ ‘                 – the blank character
```

An end-of-line character has no graphical form and is thus denoted as ‘\n’. The character ‘\’ is a so-called escape character the implies that the next character defines a special character. The following are examples of characters defined using the escape character

```
‘\n’  – end-of-line
‘\t’  – tab
‘\”   – the quote character ‘
…
```

The following example shows the declaration of data-items of type char and assigment statements:

```
ch1: val ‘!’
ch2: var Char
ch2 := ‘Q’
```

The data-item `ch1` is a a constant holding the exclamation mark character `'!'`. The data-item `ch2` is a variable of type `char`. It may hold different character values during the program execution. Execution of the assignment statement `ch2 := 'Q'` has the effect that `ch2` holds the character `'Q'`.

String values

As mention, owner is a data-item that may hold string values. A string value is a sequence of characters as opposed to char values, which are single characters. A string value may thus consist of zero or more characters. In the program text, a string value is enclosed in double quotes (“) and special characters are preceded by a slash (\). Example are shown below:

```
"John Smith"
"Hello world\n".          -- a newline is represented by \n
"a + b * (c +111)"
"He said: \"Go away!\""   -- the character " is represented by \" in the
string
```

References