5.3 String values

In the first version of class `Account`, the owner of the account was represented by a `String` data item.

```
class Account(owner: var String):
   balance: var float
   interestRate: var float
   -"-
```

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

The type `String` 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.

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 \"
```

The method `length` of a `String` returns the number of characters in the `String`, and the method `get` returns the character at a given position in a `String`.

The following example shows how to get the characters in a given `String` and print them on the console. `s: val "Hello world"` specifies that `s` has the value `"Hello world"` during the whole program execution.

```
s: val "Hello world"
for (1):to S.length :repeat
   console.print(S.get[inx])
```

The get method

The next example shows how to search for an occurrence of a given character in a `String`. The method assumes that the `aName` holds a name consisting of a first name and a surname, separated by blanks

```
aName: val "John Smith"
firstSurname:
   i: var integer
   i := 1
```

```
   while (aName.get[i] <> ' ') :repeat
       console.print(aName.get[i])
       i := i + 1
```

Here we use a while-loop which executes a sequence of statements as long as the condition is `True`.

The expression `aName.get[i] <> ' '` is the condition that is evaluated before each iteration of the loop. As long as this condition is true, the statements

```
console.print(aName.get[i])
i := i + 1
```

are executed. The operator `<>` means “not equal”, which means that the expression `aName.get[i] <> ' '` is true if `aName.get[i]` is not equal to the character `' '` (blank).

As can been seen, `i = 1` in the first iteration and the condition is thus `aName.get[1] <> ' '`, which evaluates to `true` since `aName[1] = 'J'` (the first char in `aName`) which is not equal to the the char `' '`.

In the second iteration, `i = 2` and the condition again evaluates to `true` since `aName[2] = 'o'`.

Finally when `i = 5`, the condition evaluates to `False`, since `aName[5] = ' '`, and the while-loop terminates.

String operators

A number of operators are available on `String` values.

The operator `'+'` concatenates two `String` values and return a new `String` value. This is illustrated by the following example:

```
S1,S2: var String
S1 := "John" + " Smith"
S2 := "Hello " + S1
```

The expression `"John" + " Smith"` evaluates to the `String` `"John Smith"`, which is then assigned to `S1`.

The expression `"Hello " + S1` evaluates to the `String` `"Hello John Smith"`, which is assigned to `S2`.

The operator `'='` compares two `String` values, and evaluates to `true` if the two `Strings` are identical and `false` if they differ. Consider the following example:

```
S1,S2: var String
S1 := "John"
S2 := "Liza"
S1 = S2  -- false
S1 <> S2 -- true
S1 = "John" -- true
S1 = "john" -- false
```

Note that `S1 = "john"` evaluates to `false` since the upper-case letter `'J'` differs from the lower-

case letter `'j'`.

The operator `'<='` compares two `String` values according to the numerical value of the characters in the strings.

Immutability

A `String` value is immutable, which means that it cannot be changed. There is thus no `put` method on `String` values.