The ARRAY & STRING classes in Eiffel

The ARRAY and STRING classes offer 'manifest values' which are recognised by the Eiffel system. This makes these clases more part of the language than other classes.

Manifest ARRAY

For ARRAY, the manifest form is <<it_1, it_2, it_3, ...>>

```
e.g.        famous : ARRAY[REAL]
                ...
            famous := <<3.141, 2. 718, 1.618>>
```

Manifest STRING

For STRING, the manifest form is "Any Text"

```
e.g.    s : STRING
            ...
        s := "example 1"   -- ISE Eiffel
```

Note:

ISE Eiffel does not support 'array or string constants'
e.g. s : STRING is "example 1" -- Eiffel/S only
but we can have constants of the Basic classes

```
e.g.    pi : REAL is 22/7
```

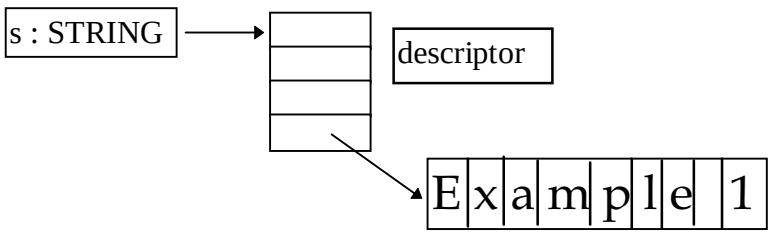
Eiffel does not regard a STRING as an ARRAY of CHARACTER because different features are associated with strings, e.g. we can append a string to another string.

Implementation:

Arrays and strings are implemented similarly.

```
If      s : STRING
            ...
        s := "example 1"
```

then the entity, s, is a reference to a 'descriptor' which may contain information about the string (its length, bounds etc) and it is the descriptor which references the the sequence of characters.



Assignment and Copy

An assignment of arrays such as

A := B

will attach the reference A to the descriptor in B and so in effect reference the same array. The sequence of values are not copied.

To copy the values of B into an array A we should use

A.copy(B)

or

A := clone(B)

The entities in the sequence B will be copied item by item into the sequence for A.

Equality of Arrays and Strings

The feature "is_equal" is redefined so that

A.is_equal(B) is true
when A is equal, item by item, to B.

Eiffel allows "A.is_equal(B)" to be written as "equal(A,B)"

Dynamic Array.

The array in Eiffel can used as a dynamic array in that it can be resized at run time. In particular a string is resized if another string is appended to it

e.g. S : STRING is "Hello "
T : STRING is "world "
then after the command

S.append(T)

S will be resized to contain the string "Hello world".

In particular an item can be 'forced' into an array so that A.force(x,k) will put item at position k, even if k is outside the bounds of the array; the array A will be appropriately resized.

Resizing is not recommended in practise as it may involve copying the whole array.

Infix @

As a concession to traditional imperative languages we can rewrite

A.item(k) as A @ k

but Eiffel recommends the "A.item(k)" notation.

The infix @ has the highest precedence and so A.item(k+1) in infix form should be A@ (k+1).

Note:

The index type of an array and string is always the integer type. Eiffel does not have 'enumeration types', but it has the alternative of 'anonymous' integers.

red, orange, green : INTEGER **is unique**

red, blue and green will be assigned different integers.

The ARRAY[G] class

The generic class ARRAY[G] has, among many others, the following features.

Creation Routine, Make

```
make (minindex, maxindex: INTEGER  )

--Allocate array;
--set index interval to -- minindex .. maxindex;
--set all values to default.
-- (Make array empty if minindex = maxindex + 1).

require
        valid_indices:
            minindex <= maxindex or (minindex = maxindex + 1)

ensure
        lower = minindex  and   upper = maxindex
```

Note:

The default values for the NUMERIC classes (INTEGER and REAL) is zero. For the class BOOLEAN the default value is **false**. For the CHARACTER class the default is the NUL character (%U). For non-Basic classes, the default value is **void**.

The Function, item

```
frozen item (i: INTEGER  ): G
        Entry at index i, if in index interval
        require
                valid_key: valid_index (k)
```

The function is 'frozen', i.e. it cannot be redefined.

The Procedure, put

```
frozen put (v: like item; i: INTEGER  )
Replace i-th entry, if in index interval, by v.

        require
                valid_key: valid_index (k)

        ensure
                insertion_done: item (k) = v
```

Note:

The argument v has the same type as item.

```
force (v: like item; i: INTEGER  )

        --Assign item v to i-th entry.
        --Always applicable: resize the array if i falls out of currently defined bounds;
        --  preserve existing items.

        ensure

                inserted: item (i) = v;
                higher_count: count >= old count
```

Note:

The procedure **force** may entail copying all the current array object.

```
has (v: G): BOOLEAN
        Does v appear in array?

        ensure

                not_found_in_empty: Result implies not empty
```

```
is_equal (other: like Current): BOOLEAN

        Is array made of the same items as other?
```

```
copy (other: like Current)

        Reinitialize by copying all the items of other.
```

count: INTEGER
Number of available indices

lower: INTEGER
Minimum index

upper: INTEGER
Maximum index

occurrences (v: G): INTEGER

Number of times v appears in structure

ensure
non_negative_occurrences: Result >= 0

prune_all (v: G)

Remove all occurrences of v.

wipe_out
Make array empty.

ensure
wiped_out: empty

all_cleared: BOOLEAN

Are all items set to default values?

empty: BOOLEAN

Is structure empty?

The Eiffel Class STRING

(see Kernal Library in the Eiffel Help file)

The Eiffel runtime system must supply special support for the class STRING; because the language includes string constants ('manifest strings')

The class STRING inherits from COMPARABLE to provide lexicographical ordering of strings and from HASHABLE to provide the possibility to use strings as hash keys. In addition to making the 'deferred' features from these abstract classes effective STRING redefines the features copy and is_equal from GENERAL in a way suitable for strings.

Some features from the class STRING

The STRING class includes among others the following:

The creation routine make

```
make(n:INTEGER)
        Allocate space for n characters.

        require
                n >= 0

        ensure
                capacity = n
```

Note:

The routine, make, makes an empty string with count equal to zero. The capacity indicates the number of possible character spaces while count indicates the actual number of characters in the string.

An empty string is not the same as a 'blank string',
i.e. a string filled with blanks. To fill a string with blanks we use fill_blank.
Before a string is created it is a **void** string.

e.g. s: STRING -- s is a void string

```
!!s.make(10)  -- creates an empty string with capacity 10
s.fill_blank  -- fills s with 10 blanks
```

In the 'Short form' presentation of the STRING class, Eiffel categorises the class routines into categories such as, Access, Comparison etc.

Some Access routines:

```
has (c: CHARACTER): BOOLEAN
        Does string include c?
```

```
hash_code: INTEGER              -- Hash code value
```

```
index_of (c: CHARACTER; start: INTEGER): INTEGER
        Position of first occurrence of c at or after start; 0 if none.
        require
                start_large_enough:     start >= 1;
                start_small_enough:     start <= count
        ensure
                correct_place: Result > 0 implies item (Result) = c
```

```
item (i: INTEGER): CHARACTER
        Character at position i
```

```
infix "@" (i: INTEGER): CHARACTER
        Character at position i
```

```
substring_index (other: STRING; start: INTEGER): INTEGER
        -- Position of first occurrence of other at or after start;  0 if none.
        require
                other_nonvoid: other /= void;
                other_notempty: not other.empty;
                start_large_enough: start >= 1;
                start_small_enough: start <= count
        ensure
                 correct_place:
                Result > 0
                        implies
                substring (Result, Result + other.count - 1).is_equal (other)
```

The Comparison routines:

```
is_equal (other: like Current): BOOLEAN
        -- Is string made of same character sequence as other
        -- (possibly with a different capacity)?
```

```
infix "<" (other: like Current): BOOLEAN
        Is string lexicographically lower than other?
```

Note:

The other comparison routines (e.g. >) are defined in terms of these two.

Some Conversion Routines:

```
to_upper
        Convert to upper case.
```

```
to_lower
        Convert to lower case.
```

```
to_integer: INTEGER
        Integer value;

        for example, when applied to "123", will yield 123
```

```
to_real: REAL
        Real value; for example,

        when applied to "123.0", will yield 123.0
```

```
mirror  -- procedure
        Reverse the order of characters.

        e.g. "Hello world" -> "dlrow olleH".

         ensure
                same_count: count = old count
```

```
mirrored: like Current          -- function
        Mirror image of string;

        result for “Hello world” is “dlrow olleH”.

        ensure
                same_count: Result.count = count
```

Note:

The routine, to_integer, converts a string to an integer. To convert an integer, k say, to a string we use the INTEGER routine, out, so that k.out is the string/printable form of the integer k.

There is no routine in the STRING class for sorting the characters into order.

The procedure mirror is such that s.mirror reverses the string s.
The function mirrored is such that s.mirrored returns the reverse of s; the string s is not changed.

Duplication Routines:

```
multiply (n: INTEGER)
        Duplicate a string within itself -- (“hello”).multiply(3) => “hellohellohello”

        require
                meaningful_multiplier: n >= 1
```

```
substring (n1, n2: INTEGER): like Current
        Copy of substring containing all characters -- at indices between n1 and n2

        require
                meaningful_origin: 1 <= n1;
                meaningful_interval: n1 <= n2;
                meaningful_end: n2 <= count

        ensure
                new_result_count: Result.count = n2 - n1 + 1
```

Note:

s.copy(t) copies the string t onto s.
Using the routine substring, s.substring(s.lower,s.upper) returns a copy of s.

Some Element Change routines:

```
append (t: STRING)
        Append a copy of t at end.
         require
                argument_not_void: t /= void

        ensure
                new_count: count = old count + t.count
```

e.g. s := "Hello "
s.append("World") is such that s is changed to refer to the string "Hello World"

```
append_character (c: CHARACTER)
        Append c at end.

        ensure
                item_inserted: item (count) = c
```

```
append_string (s: STRING)
        Append a copy of s, if not void, at end.
```

```
copy (other: like Current)
        Reinitialize by copying the characters of other.

         ensure
                new_result_count: count = other.count
```

```
extend (c: CHARACTER) -- same as append_character
        Append c at end.

        ensure
                item_inserted: item (count) = c
```

```
fill_blank      --Fill with blanks.
```

```
fill_character (c: CHARACTER)  --Fill with c.
```

```
head (n: INTEGER)
        Remove all characters except for the first n;  do nothing if n >= count.
        require
                non_negative_argument: n >= 0

        ensure
                new_count: count = n.min (old count)
```

```
insert (s: like Current; i: INTEGER)
        Add s to the left of position i in current string.
        require
                string_exists: s /= void;
                index_small_enough: i <= count;
                index_large_enough: i > 0

        ensure
                new_count: count = old count + s.count
```

```
left_adjust     -- Remove leading whitespace.

        ensure
                new_count:   (count /= 0)  implies
                        ((item (1) /= ' ') and (item (1) /= '%T')
                        and (item (1) /= '%R') and (item (1) /= '%N'))
```

```
precede (c: CHARACTER)   -- Add c at front.
        ensure
                new_count: count = old count + 1
```

```
prepend_character (c: CHARACTER)  -- same as precede
        Prepend (the string representation of) c at front.
```

```
prepend (s: STRING)        --  Prepend a copy of (not void) s at front.
        ensure
                new_count:  count = old count + s.count
```

```
prepend_string (s: STRING)
        --Same as prepend.  Prepend a copy of s (≠void), at front.
```

```
put (c: CHARACTER; i: INTEGER)
        -- Replace/overwrite character at position i by c.
```

```
replace_substring (s: like Current;  start_pos, end_pos: INTEGER)
        -- Copy the characters of s to positions   start_pos .. end_pos.
        require
                string_exists: s /= void;
                index_small_enough: end_pos <= count;
                order_respected: start_pos <= end_pos;
                index_large_enough: start_pos > 0

        ensure
                new_count:  count = old count + s.count - end_pos + start_pos - 1
```

```
replace_substring_all (original, new: like Current)
        Replace every occurence of original with new.
         require
                original_exists:        original /= void;
                new_exists:             new /= void;
                original_not_empty: not original.empty;
                not_empty:              not empty
```

```
right_adjust   -- Remove trailing whitespace.
        ensure
                new_count:   (count /= 0)   implies
                             ((item (count) /= ' ') and (item (count) /= '%T') and
                             (item (count) /= '%R') and (item (count) /= '%N'))
```

```
set (t: like Current; n1, n2: INTEGER)
        Set current string to substring of t from indices n1 to n2,
        or to empty string if no such substring.
         require
                argument_not_void: t /= void

        ensure
                is_substring: is_equal (t.substring (n1, n2))
```

```
tail (n: INTEGER)  -- see head above
        Remove all characters except for the last n;  do nothing if n >= count.
        ensure
                new_count: count = n.min (old count)
```

Note:

Some routines have more than one name.
Some routines may cause a 'forcing' of the string, i.e. the string is dynamic in size.

Some Initialization routines:

```
make_from_string (s: STRING)
        Initialize from the characters of s.
        -- (Useful in proper descendants of class STRING,
        -- to initialize a string-like object from a manifest string.)
        require
                string_exists: s /= void
        ensure
                shared_implementation: shared_with (s)
```

```
remake (n: INTEGER)  -- Allocate space for at least n characters.
         require
                non_negative_size: n >= 0
        ensure
                empty_string: count = 0;
                area_allocated: capacity >= n
```

Measurement routines:

```
capacity: INTEGER --  Allocated space
```

```
count: INTEGER      -- Actual number of characters making up the string
```

```
occurrences (c: CHARACTER): INTEGER --Number of times c appears in the string
```

Some Removal routines:

```
prune (c: CHARACTER)    -- Remove first occurrence of c, if any.
```

```
prune_all (c: CHARACTER)        -- Remove all occurrences of c.
        ensure
                changed_count:   count = (old count) - (old occurrences )
```

```
prune_all_leading (c: CHARACTER)      -- Remove all leading occurrences of c.
prune_all_trailing (c: CHARACTER)     -- Remove all trailing occurrences of c.
```

```
remove (i: INTEGER)       -- Remove i-th character.
        require
                -- 0 <= i <= count
                index_small_enough: i <= count;
                index_large_enough: i > 0
        ensure
                new_count: count = old count - 1
```

```
wipe_out      -- Remove all characters.
        ensure
                empty_string: count = 0;
                empty_area: capacity = 0
```

Resizing Routines:

```
grow (newsize: INTEGER) -- Ensure that the capacity is at least newsize.
        require
                new_size_non_negative:   newsize >= 0
```

```
resize (newsize: INTEGER)  -- see routine grow
        --Rearrange string so that it can accommodate at least newsize characters.
        --Do not lose any previously entered character.
         require
                new_size_non_negative: newsize >= 0
```