

# Exercise

- Provide an English-language description of the expected behavior of a routine  $Y = \text{SORT}(X)$  where  $X$  is an input sequence of integers and  $Y$  is an output sequence of integers
  - Assume that you are sorting in ascending order

# SORT in English

- The output vector  $\mathbb{Y}$  must be ordered
- The contents of  $\mathbb{Y}$  must be the "same as" the contents of  $\mathbb{X}$ 
  - Everything in  $\mathbb{X}$  must be in  $\mathbb{Y}$
  - Everything in  $\mathbb{Y}$  must have come from  $\mathbb{X}$
  - The number of occurrences of each item in  $\mathbb{Y}$  must be the same as its number of occurrences in  $\mathbb{X}$

# Exercise

- Now express the same behavior in predicate logic (OCL)

# Process

1. Signatures
2. Preconditions
3. Postconditions

# 1. Signatures

- You can start the process of formalization by giving the signature for the function (or relation) that you are defining
- A *signature* includes the name of the function, the type of the value that it returns and the types of its parameters

# SORT'S Signature

- The name of the function is SORT
- It returns a Sequence of Integers
- It takes a single argument of type Sequence of Integers
- So, in OCL the signature looks like

```
SORT (X : Sequence (Integer) ) :  
Sequence (Integer)
```

## 2. Preconditions

- The next step to formalization is to determine what the function's preconditions are
- A *precondition* states what a function requires to be true in order to produce its results

# SORT's Preconditions

- For SORT, any input Sequence of Integers X qualifies. Thus, SORT is always able to run
  - Assume that a type checker detects if there are missing or mistyped arguments
- This precondition is designated as `true`  
**pre:** `true`
- It can also be left out entirely

# 3. Postconditions

- The third step to formalization is to specify the function's postconditions
- A *postconditions* states what must be true after the function has completed its work
- For pure functions, a postcondition describes the computed result in terms of the input parameters
- For impure functions (procedures), you should also indicate side effects
  - Changes to global variables
  - Input/output
- For OO methods, changes to instance variables must also be described

# SORT's Postcondition

- SORT's output has two properties
  - It must be in order
  - The elements must be the same as the input's
- We can make use of OCL's `Bag` class  
**post:** `ORDERED(Y) and Y.asBag() = X.asBag()`
- We will take another approach using permutations  
**post:** `ORDERED(Y) and PERMUTATION(X, Y)`

# ORDERED

- Now give a formal specification for ORDERED
  - Signature
  - Precondition
  - Postcondition

## ORDERED - 2

- Now give a formal specification for  
ORDERED

- Signature

ORDERED (Y : SEQUENCE Integer) : Boolean

- Precondition

- Postcondition

## ORDERED - 2

- Now give a formal specification for ORDERED
  - Signature

```
ORDERED (Y : SEQUENCE Integer) : Boolean
```
  - Precondition

```
pre: true
```
  - Postcondition

# ORDERED Postcondition

- In predicate logic we have

$$\forall i : 1 \leq i < |Y| \bullet Y[i] \leq Y[i + 1]$$

– *For all positions in Y from the first through the one before the last, the value stored in the sequence at that position is no greater than the value stored at the next position*

- Sequences have an indexing operation
- Indexing of Sequences starts from 1
- $\forall$  reads "for all"
- $|Y|$  is the length of Y

# ORDERED Postcondition OCL

```
Y->forall (e1 : Integer |
```

```
  Y->forall (e2 : Integer |  
    indexOf (e1) < indexOf (e2)  
    implies e1 <= e2))
```

- -> **navigates from a Collection**
- `forall` is an operation of Collections
- **First comes a bound variable `e1`**
- **Then a Boolean expression**
- **The overall result is `true` if the Boolean expression is `true` when the bound variable takes on all possible values from the Collection**

# PERMUTATION

- Now give a formal specification for PERMUTATION
  - Signature
  - Precondition
  - Postcondition

# PERMUTATION

- Now give a formal specification for PERMUTATION

- Signature

```
PERMUTATION (X : Sequence(Integer),  
            Y : Sequence(Integer)) : Boolean
```

- Precondition

- Postcondition

# PERMUTATION

- Now give a formal specification for PERMUTATION

- Signature

```
PERMUTATION (X : Sequence(Integer),  
            Y : Sequence(Integer)) : Boolean
```

- Precondition

```
pre: X->size() = Y->size()
```

- Postcondition

# PERMUTATION Postcondition

**def:** `s = X->size()`

`(s = 0) or`

**let**

`Integer j = Y->indexOf(X->first())`

**in**

`PERMUTATION(X->subsequence(2, s),`

`(Y->subsequence(1, j-1)) ->`

`union(y->subsequence(j+1, s))))`