

Sort Exercise

- Sorting is a well-understood concept that is surprisingly hard to specify precisely
- For this exercise, try to formally specify a `SORT` routine that takes a vector of integer values as input and returns a sorted vector of integer values as output
 - You may assume the routine sorts in ascending order
- To do this, try the following steps:
 1. First give a signature for the routine
 2. Then try to describe the relation between the input and the output in English
 3. Now try to specify the relation in predicate calculus use pre and post conditions

1. Signature

- `Vector<int> Y = SORT (Vector<int> X)`
 - `X` and `Y` are explicitly named so we can use them in the pre and post conditions
 - `int` could actually be any data type which supports a comparison (`>`) operator
- **Alternatively, a relational form could be used**
 - `SORT (Vector<int> X, Vector<int> Y)`

2. English Specification

- ?

Trouble Spots

- What does your specification do in the following situations?

– $\langle 1, 2 \rangle \rightarrow \langle 4, 5 \rangle$

– $\langle 1, 1, 2, 3 \rangle \rightarrow \langle 1, 2, 3 \rangle$

– $\langle 1, 2, 3 \rangle \rightarrow \langle 1, 1, 2, 3 \rangle$

– $\langle 1, 3, 3, 2, 2, 2 \rangle \rightarrow$

$\langle 1, 2, 2, 3, 3, 3 \rangle$

2. English Specification

- Y is an ordered version of X
 - Y is ordered
 - Y and X have the same elements
 - With the same numbers of occurrences!

3. More Precise Specification

- Now express the same behavior more precisely
 - a. First give a signature (parameter and return types)
 - Should be the same as what you have already done
 - b. Next determine the situations in which you expect the program to work (its preconditions)
 - c. Then try to describe the relation between the input and the output in English (postconditions)
 - d. Finally, translate your specifications into predicate calculus

b. Preconditions

- ?

b. Preconditions

- For `SORT` any input vector of integers `X` qualifies
 - The preconditions assume the types are correct
- Thus, `SORT` is always able to run
 - This precondition is designated as `true`

c. Postconditions

- ?

c. Postconditions

1. Y must be the same length as X

c. Postconditions

1. Y must be the same length as X
2. The output vector Y must be ordered

c. Postconditions

1. Y must be the same length as X
2. The output vector Y must be ordered
3. The contents of Y must be the "same as" the contents of X

c. Postconditions

1. Y must be the same length as X
2. The output vector Y must be ordered
3. The contents of Y must be the "same as" the contents of X
 - Everything in X must be in Y
 - Everything in Y must have come from X
 - The number of occurrences of each item in Y must be the same as its number of occurrences in X
 - Another term used to express these properties is to say that Y is a *permutation* of X
 - Yet another way is to say that if the contents of X and Y are treated as bags, the two bags are equal
 - A *bag* is like a set, except that multiple instances of an element are allowed and that a count is kept of the occurrences

d. Predicate Calculus for SORT

Vector Y = SORT (Vector X)

pre: true

post: LENGTH (X) == LENGTH (Y) \wedge

ORDERED (Y) \wedge

PERMUTATION (X, Y)

1. Y is the Same Length as X

- X and Y are `Vectors`
- `Vectors` (as a datatype) have properties, one of which is their length
 - `|X| == |Y|`
 - `X.length() == Y.length()`
 - `#X == #Y`
- This raises the question of who is responsible for allocating the space for Y
 - Caller preallocates or `SORT` does the allocation

2. \mathbb{Y} is ordered

- Try to express this property using the predicate calculus

2. Y is ordered

- $\forall i : 1 \leq i < |Y| \bullet Y[i] \leq Y[i + 1]$
 - *For all positions in Y from the first through the one before the last, the value stored in the vector at that position is no greater than the value at the next position*
- `Vectors` have an indexing operation
- Counting starts from 1
- \forall reads "for all"

3. The Contents of \mathcal{Y} is the "same as" the Contents of \mathcal{X}

- Try to express this in the predicate calculus

3. The Contents of Y is the "same as" the Contents of X

- Two vectors with the same contents but in which the values may be ordered differently are called *permutations*
- Y is a permutation of X ; $\text{PERM}(X, Y)$
 - Precondition: $|X| == |Y|$
 - Postconditions
 - $|X| == 0 \Rightarrow \text{true}$
 - $X[1] == Y[1] \Rightarrow \text{PERM}(\text{tail}(X), \text{tail}(Y))$
 - $\exists j : 1 < j \leq |Y| \bullet X[1] == Y[j] \wedge \text{PERM}(\text{tail}(X), \text{concat}(Y[1 .. j-1], Y[j+1 .. |Y|]))$

Explanation

- \Rightarrow is an *if* expression, with the condition on the left, and the consequent on the right
- There is an implicit "else" between clauses
- `tail` is an operation on `Vectors` that returns a new `Vector` like the argument except that the first element has been removed
- `concat` is an operation on `Vectors` that creates a new `Vector` by pasting together the two `Vector` arguments
- \exists reads "there exists"
- $i \dots j$ denotes the integers from i to j inclusive

Summary

- Vector $Y = \text{SORT}(\text{Vector } X)$

pre: true

post: $\text{ORDERED}(Y) \wedge \text{PERM}(X, Y)$

– $\text{ORDERED}(Y) \Leftrightarrow (\forall i \in 1..|Y|-1 \bullet Y[i] \leq Y[i+1])$

– $\text{PERM}(X, Y) \Leftrightarrow$

$|X| = |Y| \wedge (|X| > 0 \Rightarrow$

$((X[1] = Y[1]) \wedge \text{PERM}(\text{tail}(X), \text{tail}(Y))) \vee$

$(X[1] \neq Y[1]) \wedge$

$(\exists j: 1 < j \leq |Y| \bullet (X[1] = Y[j]) \wedge$

$\text{PERM}(\text{tail}(X), (Y[1..j-1] \hat{\cap} Y[j+1..|Y|])))$