OOP Cristea Andrei: Documentation, 2'nd assignment.

Cristea Andrei NEPTUN CODE	**2 assignment. Task 9**	01.04.2021
w61rab@inf.elte.hu Group 4		

Task

At every competition of the National Angling Championship, the results of the competitors were recorded and put into a text file. Every line of the file contains the name of the angler, the ID of the competition (string without spaces), and the species and the size of the caught fishes (pair of a string without spaces and a natural number). Data is separated by space or tab. The lines of the file are ordered by the name of the anglers. The file is assumed to be in the correct form. Sample line in the file:

Peter LAC0512 carp 45 carp 53 catfish 96

(1) Give the average number of caught catfishes. If no catfish is caught, the average is zero. (2) Name the angler who has caught the biggest number of catfishes.

(1) First part

Plan of main program

A = (f: infile(Line), elem: CaughtItem, count: N, result: N)

Line = rec(name: String, catches: $Catch^*$)

Catch = rec(first: String, second: Z)

CaughtItem= rec(name: String, countOfItems: N)

New state space:

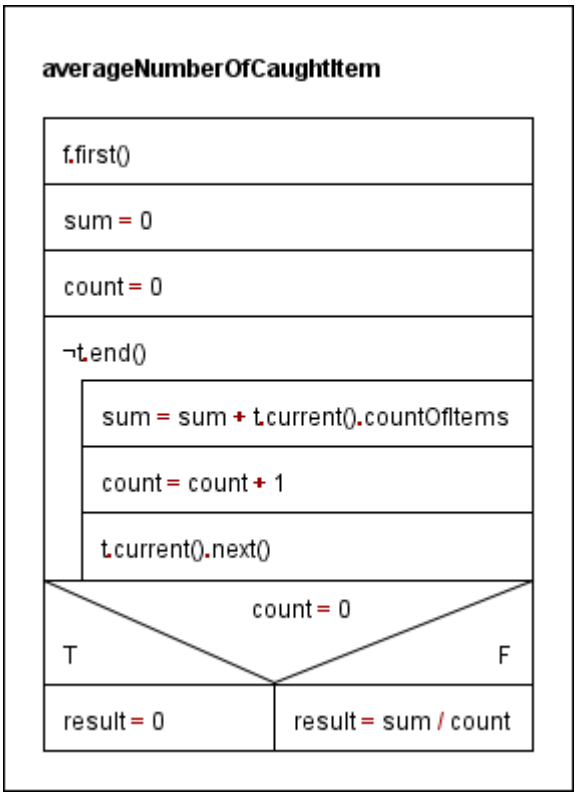
A = (t = enor(CaughtItem), count: N, sum: N result: Z)

Pre = (t = t')

Post =

$$count = \Sigma_{e \in t'} 1 \wedge sum = \Sigma_{e \in t'} e..current().countOfItems \wedge$$

$$((count > 0 \wedge result = sum / count) \vee (count = 0 \wedge result = 0))$$



Analogy: Summation, Custom Enumerator

E	~ CaughtItem
cond(E)	~ true

Enumerator of $CaughtCustomEnumerator$ [1]

$enor(CaughtItem)$	$first()$, $next()$, $current()$, $end()$
f: $infile(Line)$	$first()$ ~ $next()$
cur: $CaughtItem$	$next()$ ~ $see\ below$
end: L	$current()$ ~ $return\ cur$
	$end()$ ~ $return\ end$

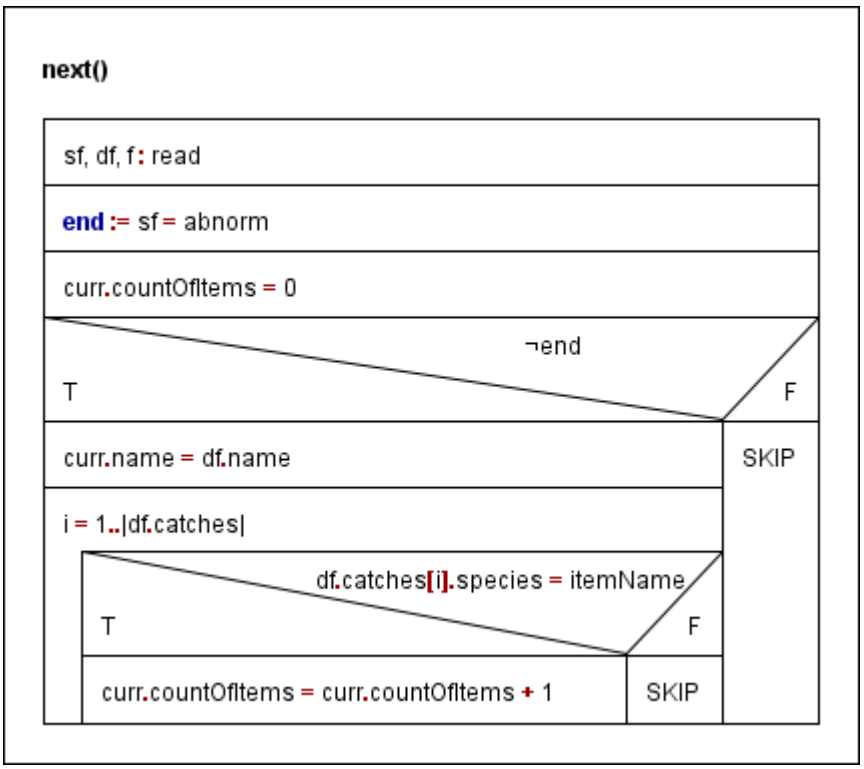
$Status = \{norm, subnorm\}$

In enor(CaughtCustomEnumerator), operations first() and next() have the same purpose, to read the line line of text stream(f sequential input file). If there is no more available line, then the end function returns true, meaning the end of enumeration. If there are some then anglers name and the sequence of its haul will be obtained in the next() function call. We will add to countOfItems field new values if only the catches species at current state will have the same fish name as passed by itemName(current case catfish)

$A^{next} = (f: infile(Line), cur: CaughtItem, end: L, itemName: String)$

$Pre^{next} = (f = f' \wedge itemName = itemName')$

$Post^{next} = (itemName = itemName') \wedge (sf, df, f = read(f')$
$\wedge (end = (sf = abnorm)) \wedge (\neg end \rightarrow curr.name = df.name$
$\wedge curr.countOfItems = (\Sigma_{i \in [1..|df.catches|],\ df.catches[i].species == itemName} 1)$



Analogy: Counting, On interval enumerator

t[m..n]	~ i : [1 .. |df.catches|]|
cond(i)	~ df.catches[i].species = itemName
c	~ curr.countOfItems

(2) Second part
Plan of main program

A = (f: infile(Line),name: String, number: N)

Line = rec(name: String, catches: $Catch^*$)

Catch = rec(first: String, second: Z)

New state space:

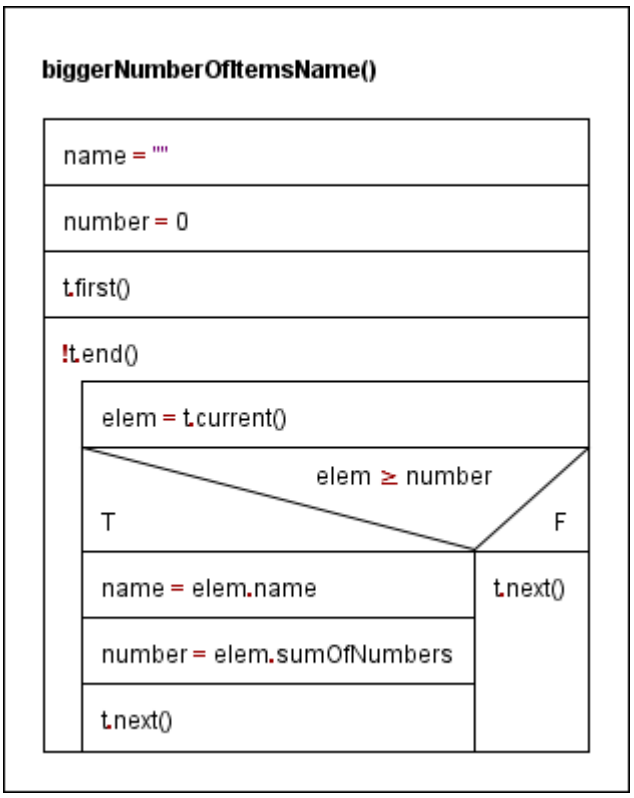
A = (t = enor(NameAngler), name: String, number: N, elem: NameAngler)

Pre = (t = t' ∧ $name = name'$ ∧ $number = number'$)

Post =

$(number,\ elem)\ =\ MAX_{e=t'}\ e.current().sumOfItems\ \geq\ number$

$\wedge\ name\ =\ elem.name$



Enumerator of $NameAnglerEnumerator$ [2]

$enor(NameAngler)$	$first(),\ next(),\ current(),\ end()$
tt: $enor(CaughtItem)$	$first()$ ~ $tt.first(),\ next()$
cur: $NameAngler$	$next()$ ~ *see below*
end: L	$current()$ ~ *return cur*
	$end()$ ~ *return end*

$CaughtItem\ =\ rec(name\colon String,\ countOfItems\colon N)$

The enor(CaughtItem) and next function should solve the following task. Start a new CaughtItem enumerator in first() function call and then in the next() as tt.current() has summation attribute from the string we pass into it (in our case in it is “catfish”), and we iterate over it while while name in tt.current().name is same as in the t.current().name while the tt.end() is not equal to true. It will look this way in specification:

$$A^{next} = (tt\colon enor(CaughtItem),\ curr\colon NameAngler\ ,\ end\colon L\ )$$

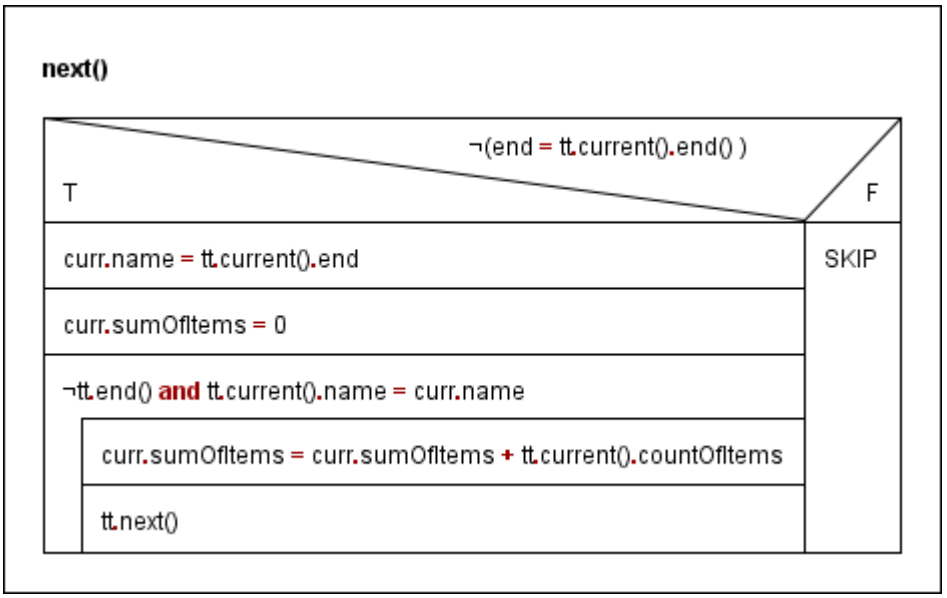
$$Pre^{next} = (tt = t''\ )\ .$$

$$Post^{next} = \neg end = tt.current().end() \ -> \ curr.name = tt.current().name \ \wedge$$

$$curr.sumOfItems = \Sigma_{\ e\ \in tt'.current(),\ e.current().name\ =\ curr.name\ \wedge\ !e.current().end()}\ e.current().countOfIte$$

Analogy: Summation, Custom enumerator

t: enor(E)	~tt’: enor(CaughtItem)
e ∈t’	e∈t’’.current()
cond(i)	e.current().name = curr.name
c	~ curr.sumOfItems



Testing

There are three algorithmic patterns which are used in this project: Summation, Counting and Maximum Selection

A. Summation in the first part => Give the average number of caught catfishes. If no catfish is caught, the average is zero.
   length-based
   1. Enor's end() => true
   2. Enor does one iteration, after which it has end() => true
   3. Enor does more than one iteration, after which it has end() => true

   first and last-based
   1. Last one only enor.current().countOfItems > 0
   2. First one only enor.current().countOfItems > 0

   pattern-base
   1. All enor.current().countOfItems are equal to 0
   2. Only one countOfItems from Enor is not equal to 1
   3. More than one countOfItems from Enor are not equal to 0

B. Counting in first part custom enumeration (CaughtItemsEnumerator) => Find sum and countOfItems of caught catfishes by single angler
   length-based
   1. Line without catches
   2. Line with one catch
   3. Line with more than one catches

   first and last-based
   1. Line with catches, first one is catfish
   2. Line with catches, last one is catfish

   pattern-base
   1. Line without catfishes
   2. Line with one catfish
   3. Line with more than one catfishes

C. Summation in first part enumeration (CaughtItemsEnumerator) => Find a sum and countOfItems of caught catfishes by single angler

   Since Counting and Summation patterns use the same logic making a new separate Test checking is redundant

D. Maximum Search in second part (main program) => Name the angler who has caught the biggest number of catfishes.
   length-based
   1. NameAngleEnumerator end() = true
   2. NameAngleEnumerator will iterate over one item and then end() = true
   3. NameAngleEnumerator will iterate over more than one item and then end() = true

   first and last-based
   1. Maximum value of current().sum will be in a first call to NameAngleEnumerator
   2. Maximum value of current().sum will be in a the last call to NameAngleEnumerator

   pattern-base
   1. Maximum value = 0 for all items in enumeration
   2. Maximum value will have only one item in enumeration
   3. Maximum value will have more than one item in enumeration

E. Summation in second part (NameAngleEnumerator) => Summing all summed with CaughtItemsEnumerator catfishes amounts, and doing it while name is the same of the next angler
   length-based
   1. CaughtItemsEnumerator has no items to enumerate
   2. CaughtItemsEnumerator has one to enumerate
   3. CaughtItemsEnumerator has multiple to enumerate

   first and last-based
   1. In next() enumeration only last row from CaughtItemsEnumerator has catfishes

2. In next() enumeration only first row from CaughtItemsEnumerator has catfishes

pattern-base

1. All items are placed in one row
2. Items are placed in several rows