Teréz A. Várkonyi **2. assignment/0. task** 11th February 2019
NEPTUNCODE
nick@inf.elte.hu
Group 0

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 an angler who caught at least two catfishes on of the competitions. If there is any, write out his name, the ID of the competition and the number of catfishes he has caught.

(2) Name an angler who caught at least two catfishes on each of his competition.

(1) First part

Plan of the main program:

A = (f : infile(Line), l : $\mathbb{L}$, elem: Contest)
Line = rec (angler : String, contest: String, catches : Catch)*
Catch = rec (species : String, size : $\mathbb{N}$)
Contest = rec (angler : String, contest : String, counter : $\mathbb{N}$)

New state space:

A = (t : enor(Contest), l : $\mathbb{L}$, elem : Contest)
Pre =(t = t')
Post =($l, elem = \underset{e \in t'}{\boldsymbol{SEARCH}}(e.counter \geq 2)$)

```
l:=false;        t.first()
¬l ∧ ¬t.end()
    l, elem := t.current().counter ≥ 2, t.current()
    t.next()
```

Analogy: linear search, custom enumerator

E	~ Contest
cond(e)	~ e.counter ≥ 2

Enumerator of Contests[1]

enor(Contest)	*first(), next(), current(), end()*

f : *infile*(*Line*)
cur : *Contest*
end : $\mathbb{L}$
	first() ~ *next()*
	next() ~ *see below*
	current()~ **return** *cur*
	end() ~ **return** *end*

Status = { norm, abnorm }

In enor(Contest), operations first() and next() are the same. They have to solve the following task: read the next line of the textfile (f sequential input file). If there is no more, then vaiable end gets true. If there is any, the current angler’s name and the contest ID can be extracted. Then, the ”catfish” species can be counted in the catches..

A^{next} = (*f: infile(Line), cur : Contest, end :* $\mathbb{L}$)
Pre^{next} = (*f = f’*)
$Post^{next}$ = (*sf, df, f = read(f’)* $\wedge$ *end = (sf=abnorm)*
$\wedge \neg$*end* $\rightarrow$ *cur.angler = df.angler* $\wedge$ *cur.contest = df.contest*
$\wedge$ *cur.counter =* $\sum_{\substack{i\in[1..|df.catches|] \\ df.catches[i].species="catfish"}} 1$)

```
sf, df, f : read
end := sf=abnorm
¬end
    cur.angler,  cur.contest := df.angler, df.contest
    cur.counter := 0
    i = 1 .. |df.catches|
        df.catches[i].species = ”catfish”
            cur.counter := cur.counter + 1
        else
            SKIP
else
    SKIP
```

Analogy: counting, on interval enumerator

t: [m .. n]	~ i : [1 .. |df.catches|]
cond(i)	~ df.catches[i].species=”catfish”
c	~ cur.counter

[1] The enumerator should be in a separate compilation unit.

(2) Second part

Plan of the main program

A = (f : infile(Line), l : $\mathbb{L}$, id : String)
Line = rec (angler : String, contest : String, catches : Catch)*
Catch = rec (species : String, size : $\mathbb{N}$)

New state space:

A = (t : enor(Angler), l : $\mathbb{L}$, id : String)
Angler = rec (id:String, skillful:$\mathbb{L}$)
Pre = (t = t')
Post = ($l, elem = \underset{e \in t'}{\boldsymbol{SEARCH}}(e.skillful) \wedge l \rightarrow id = elem.id$)

```
l:=false;   t.first()
¬l ∧ ¬t.end()
    l, elem := t.current().skillful, t.current()
    t.next()
l
    id := elem.id  |  SKIP
```

Analogy: linear search, custom enumerator

E	~ Angler
cond(e)	~ e.skillful

Enumerator of Anglers[2]

enor(Angler)	*first(), next(), current(), end()*	
tt : *enor*(*Contest*)	*first()*	~ *tt.first(); next()*
cur : *Angler*	*next()*	~ *see below*
end : $\mathbb{L}$	*current()*	~ **return** *cur*
	end()	~ **return** *end*

Contest = rec (angler : String, contest : String, counter : $\mathbb{N}$)

In *enor(Angler),* operation *next()* has to solve the following task:
Give the next angler and decide if he has caught 2 catfishes on each of his contest. Tod o the decision, the enumerator of the first part can be used (*enor(Contest)*) which processes one

---

[2] The enumerator of Anglers, too, should be in a separate compilation unit.

line of the input file and calculates how many catfishes the angler has caught on one contest. At the beginning of operation *next()*, the current item of the Contest enumerator is the first contest of the angler to be processed, so for getting *tt.current(), tt.first()* and *tt.next()* are not needed. The enumeration runs as long as the same angler's contests are "read" with *tt.next()*.

A^{next} = (*tt : enor(Contest) , cur : Angler, end :* $\mathbb{L}$)
Pre^{next} = (*tt = tt'*)
$Post^{next}$ = ($end = tt'.end() \land \neg end \rightarrow (cur.id = tt'.current().angler \land$

$$(cur.skillful, tt) = \bigwedge_{e \in (tt'.current(), tt')}^{e.angler = cur.id} (e.counter \geq 2))$$

next()

```
end := tt.end()
if ¬end then
    cur.id := tt.current().angler
    cur.skillful := true
    while ¬tt.end() ∧ tt.current().angler = cur.id
        cur.skillful := cur.skillful ∧ tt.current().counter ≥ 2
        tt.next()
else
    SKIP
```

Analogy: summation (optimistic linear search), custom enumerator

t:enor(E)	~ tt:enor(Contest)
	without first
	as long as e.angler = cur.id
eϵt'	~ e ϵ (tt'.current(),tt')
s	~ cur.skillful
(H,+,0)	~ ($\mathbb{L}$, ∧, true)

Testing

Three algorithmic patterns are used in the solution: linear search, optimistic linear search, and counting.

A. Linear search in the first part:
   – Searching for one angler who has caught at least 2 catfishes. **–**
   length-based:
   1. Empty file.
   2. One angler.
   3. More anglers.

   first and last-based:
   4. First angler has caught two catfishes.
   5. Only the last angler has caught two catfishes.

pattern-based:

1. There is an angler who has caught at least two catfishes.
2. There is no such angler.
3. There are more anglers who meet the requirements.

B. Number of the caught catfishes on one contest (counting)

length-based:

1. Line without catches.
2. Line with one catch.
3. Line with more catches.

first and last-based:

4. Line with catches, the first one is a catfish.
5. Line with catches, the last one is a catfish.

pattern-based:

7. Line without catfish catch.
8. Line with one catfish catch.
9. Line with more catfish catches.

C. Linear search in the second part:

– Searching for a skillful angler (who has caught at least two catfishes on each of his contests). –

length-based:

6. Empty file.
7. One angler.
8. More anglers.

first and last-based:

9. First angler is skillful.
10. Only the last angler is skillful.

pattern-based:

4. There is skillful angler.
5. There is no skillful angler.
6. There are more skillful anglers.

D. Deciding if an angler is skillful (optimistic linear search):

length-based:

1. No catch.
2. One angler’s one contest.
3. One angler’s more contests.

first and last-based:

4. An angler has not caught two catfishes only on his first contest.
5. An angler has not caught two catfishes only on his last contest.

pattern-based:

6. There is not any contest where he has caught at least two catfishes.
7. He has caught at least two catfishes on only one of his (more than one) contests.
8. He has caught at least two catfishes on each of his contests.