|  |  |  |
| --- | --- | --- |
| **Aly Elalwany WGL13Q** [**wgl13q@inf.elte.hu**](mailto:wgl13q@inf.elte.hu)  **Group 3** | **2nd Assignment** | **April 2021** |

**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 contest ID. 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 first angler with the contest ID, who had at least one contest without catch.   
(2) On which contest were the most fishes caught?

1. **First Part**

**Plan of the program :**

A = ( f : infile(Angler), elem: (Angler) )   
Angler = rec (name : String,ID: String,none\_caught:Bool)  
Measurement = rec (species : String, fish\_Size : ℕ)   
Contest = rec (ID: String,sum : ℕ)

New state space :

A = ( f : infile(Angler), elem: Angler)

Pre = (t = t’)

Post = ( elem = Search (e. none\_Caught())

e belongs t’

|  |
| --- |
| t.first() |
| !t.end()  t.current().none\_Caught    elem.ID, elem.name : = t.current().ID  t.current().name;   t.next() |

Analogy : Linear Search

E ~ Angler

Cond(e) ~ e.none\_Caught

Enumerator of Angler

|  |  |
| --- | --- |
| enor (Angler) | first (),read (), current() , end() |
| f: infile(Angler)  dx : Angler status : sx  end : **L** | first() ~ read()  next() ~read()  current() ~ return dx  end() ~ return sx = abnorm |

In enor(Angler) operation first() opens the file and read() reads the next line of the text file. If there’s no more, the variable end gets true. If there’s any , the current angler’s name and ID can be extracted using the current() method.

A = (f:infile(Angler) , a: Angler, m: Measurement)

Pre = (f=f ‘)

Post = (sx,dx,x )=^ end = (sf = abnorm)

^ !end -> a.none\_Caught = a.none\_Caught ^ m.fish\_Size <=0

^ a.sum =sumOfFish = ∑ m.fish\_Size   
 a.name == dx.name

**Next()**

|  |
| --- |
| End : = sf = abnorm  sx,dx,x : read |
| a.none\_Caught := true  sumOfFish := 0  !end()   a.none\_Caught := a.none\_Caught ^ m.fish\_Size <=0  sumOfFish := sumOfFish + m.fish\_Size  a.sum := sumOfFish  sx,dx,x : read |

We are enumerating through the measurements in the file and we are adding the fish sizes and for that line where there is one contest where the angler didn’t catch any fish we leave the value none\_Caught to be true thus meaning that the angler had a contest where he caught no fish.

Analogy :Summation

f(e) ~ m.fish\_Size

s ~ a.sum

H, + , 0 ~ N , +, 0

**2) Second Part**

**Plan of the main program**

A = (f:infile(Angler) , c :Contest )

Angler = rec (name : String,ID: String,none\_caught:Bool)  
Contest = rec (ID: String,sum : ℕ)

New State Space :

A = (t:enor(Angler) , max: N )

Pre = (t = t’)

Post = (max = **Max** c.sum)  
 e € t‘

|  |
| --- |
| Max :=0 t.first() |
| !t.end()   |  | | --- | | t.current().sum > max  max := t.current().sum  c = t.current()  t.next() | |

Analogy : Max Search , custom enumerator

E ~ Angler

F(e) ~ c.sum

S ~max

Enumerator of Angler

Enumerator of Angler

|  |  |
| --- | --- |
| enor (Contest) | first (),next (), current() , end() |
| f: infile(Angler)  dx : Angler  Curr : Contest  status : sx  end : **L** | first() ~ sx,dx,x : read next()  current() ~ return cur  status : return sx  end() ~ return end |

In enor(Angler), operation next() has to solve the following task:  
On which contest were the most fishes caught?

It goes through the file as it is sorted by the contest ID we enumerate the contest and then we sum the number of fish caught in each contest and then we loop through them grabbing the contest with the max number.

|  |
| --- |
| end:= sx =abnorm |
| !end |
| sumGrp : = 0  S K I P  curr.ID : = \_dx.ID  !end ^ curr.ID = dx.ID  sumGrp := sumGrp + dx.sum   curr.sum :=sumGrp   sx,dx,x:read |

Analogy: summation  
t:enor(E) ~ t:enor(Contest) without first as long as curr.ID = dx.ID

S~ curr.sum

(H,+,0) ~ (**N**, + ,0)

**Testing**

We are using Linear Search, Max Search and Summation algorithmic patterns in this program

1. Linear search in the first part:

Searching for the angler who has caught 0 fish in a specific contest

Length-based

1.Empty File

2. One angler has caught no fish

3. One angler has caught fish thus not selected

4. More anglers outputting the first one who didn’t catch any fish at a contest

5. More anglers, third one in the group meets the requirements

6. More anglers, last one in the group meets the requirements

7. More anglers, no one in the group meets the requirements

1. Linear max search in the first part with summation included  
   Searching for the contest with the max amount of fish caught.
2. Empty file
3. Only one in the group meets requirements
4. First group’s contest in the group meets the requirements
5. Second group’s contest meets requirements
6. Last group’s contest meets requirements