5th practice

1. How many even numbers there are in a sequential input file containing integers?

a) How many even numbers there are before the first negative number in the file?

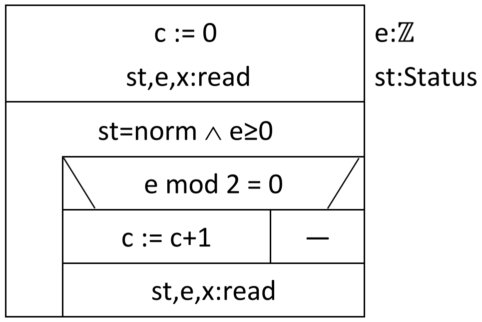
Specification:

$A = (x:infile(\mathbb{Z}), c:\mathbb{N})$

$Pre = (x=x_0)$

$Post = (c = \sum_{\substack{e \in x_0 \\ 2|e}}^{e \geq 0} 1)$

Algorithm:



Analogy: Couning

t:enor(E) ~ x:infile(ℤ) (st,e,x:read)
as long as e≥0

cond(e) ~ 2|e

Remark: In the specification, the extra condition above the keyword of the algorithmic pattern shows how long the enumeration goes.

b) How many even numbers there are after the first negative number in the file?

Specification:

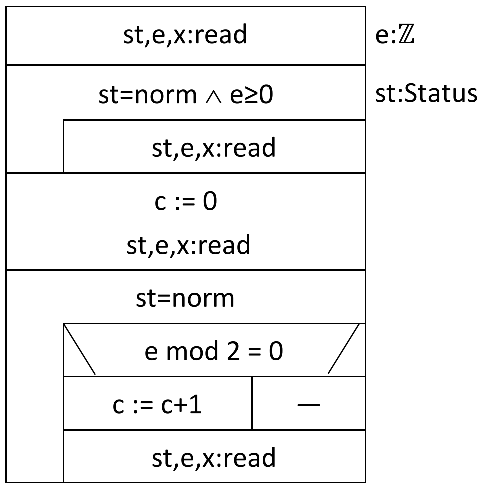
$A = (x:infile(\mathbb{Z}), c:\mathbb{N})$

$Pre = (x=x_0)$

$Post = ((e', (st',e',x')) = \textbf{SELECT}_{e \in x_0}(st=abnorm \vee e<0)$

$\wedge\ c = \sum_{\substack{e \in x' \\ 2|e}} 1)$

Algorithm:



Analogy: Selection

t:enor(E) ~ x:infile(ℤ) (st,e,x:read)

cond(e) ~ e<0 ∨ st=abnorm

Counting

t:enor(E) ~ x:infile(ℤ) „in process"
enumeration (st,e,x:read)

cond(e) ~ 2|e

Remarks:

- The Selection has two results: the selected element (denoted by e', it stores the current value of e) and the enumerator the current state of which is stored in (st',e',x').
- The specification can show three different states of the algorithm. For example, three different values of variable *x* can be seen in the post condition: x_0 is responsible for the initial value, x' stores the result of the Selection, and x denotes the final value of the enumerator, which is an empty enumeration.

c) How many even numbers there are before and after the first negative number in the file?

Specification:

A = (x:infile($\mathbb{Z}$), c1, c2:$\mathbb{N}$)

Pre = (x=x_0)

Post = ((c1, (st',e',x')) = $\sum_{\substack{e \in x_0 \\ 2|e}}^{e\geq 0} 1$

$\wedge$ c2 = $\sum_{\substack{e \in x' \\ 2|e}} 1$)

Counting

t:enor(E) ~ x:infile($\mathbb{Z}$) (st,e,x:read)
as long as e≥0

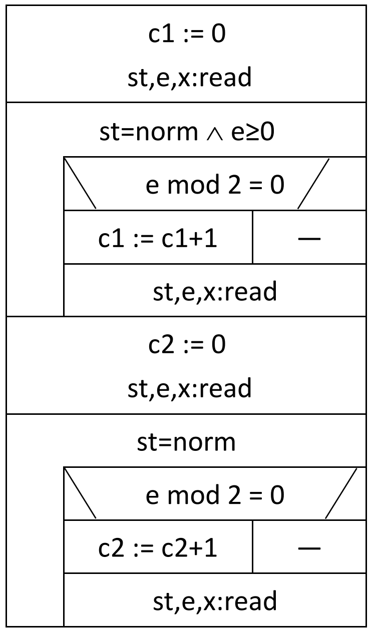
cond(e) ~ 2|e

Counting

t:enor(E) ~ x:infile($\mathbb{Z}$) (st,e,x:read)
next() instead of first()

cond(e) ~ 2|e

Algorithm:



e:$\mathbb{Z}$
st:Status

Remark: At the beginning of the second counting, instead of calling first(), operation next() is used, as the enumeration is already „in process” (luckily, for sequential input files the two operations are the same).

d) How many even numbers there are before and after the first negative number in the file if in the after version that first negative number is included, too?

Specification:

A = (x:infile($\mathbb{Z}$), c1, c2:$\mathbb{N}$)

Pre = (x=x_0)

Post = ((c1, (st',e',x')) = $\sum_{\substack{e \in x_0 \\ 2|e}}^{e\geq 0} 1$

$\wedge$ c2 = $\sum_{\substack{e \in (e',x') \\ 2|e}} 1$)

Counting

t:enor(E) ~ x:infile($\mathbb{Z}$) (st,e,x:read)
as long as e≥0

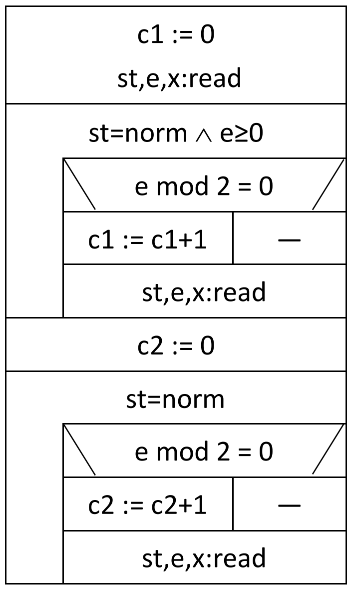
cond(e) ~ 2|e

Counting

t:enor(E) ~ x:infile($\mathbb{Z}$) (st,e,x:read)
without first()

cond(e) ~ 2|e

Algorithm:



e:$\mathbb{Z}$
st:Status

Remark: In the specification of the second counting, it has to be shown that the previously read e' is included in the enumeration (as the sequence to be enumerated was <e'>⊕x'). The enumeration does not start with operation first(), as variable *e* already stores the value of e'. That is why the reading is missing from the initialization of the second counting.

2. In a sequential input file, transactions of customers are stored. Each transaction contains the customer's ID, the transaction's date, and the amount which is a signed integer (depending on the type of the transaction: positive if it is a deposit and negative if it is e.g. a withdrawal). The file is ordered by the customers' ID. Find the first customer's transaction with the highest deposit.

Specification:

A = (x:infile(Transaction), elem:Transaction)
Transaction = rec(ID:String, date:String, am:$\mathbb{Z}$)
Pre = ($x=x_0 \wedge |x|>0 \wedge x\nearrow_{ID}$)
Post = (ID = $(x_0)_1$.ID $\wedge$
(max, elem) = $\mathbf{MAX}_{e\in x_0}^{e.ID=ID}$ e.am)

Maximum search
t:enor(E) ~ x:infile(Transaction) (st,e,x:read) as long as e.ID=ID
f(e) ~ e.am
H, < ~ $\mathbb{Z}$, <

Algorithm:

st,e,x:read	st:Status
max, elem, ID := e.am, e, e.ID	max:$\mathbb{Z}$, ID:String
st,e,x:read	e:Transaction
loop e.ID=ID $\wedge$ st=norm	
if max < e.am **then** max, elem := e.am, e, **else** —	
st,e,x:read	

3. Calculate the average of daily temperatures stored in a sequential input file.

a) Calculate the average of daily temperatures before the first temperature under the freezing point (in a sequential input file).

Specification:

A = (x:infile($\mathbb{R}$), a:$\mathbb{R}$)
Pre = ($x=x_0 \wedge |x|\geq 1 \wedge x_0[1]>0$)
Post = (a = $\sum_{e\in x_0}^{e\geq 0} e \,/\, \sum_{e\in x_0}^{e\geq 0} 1$)

Two summations
t:enor(E) ~ x:infile($\mathbb{R}$) (st,e,x:read) as long as e≥0
f(e) ~ e, 1
s ~ s, c
(H, +, 0) ~ ($\mathbb{R}$ +, 0.0), ($\mathbb{N}$ +, 0)

Algorithm:

s, c := 0.0, 0 st,e,x:read	e, s:$\mathbb{R}$, c:$\mathbb{N}$ st:Status
loop e≥0 $\wedge$ st=norm	
s, c := s+e, c+1	
st,e,x:read	
a := s / c	

b) Calculate the average of daily temperatures after the first temperature under the freezing point (in a sequential input file).

Specification:

A = (x:infile($\mathbb{R}$), a:$\mathbb{R}$)
Pre = ($x=x_0 \wedge |x|\geq 2 \wedge \exists i\in[1..|x|-1]: x_0[i]\leq 0$)
Post = ((e', x') = $\mathbf{SELECT}_{e\in x_0}$(e≤0 $\vee$ st=abnorm)
$\wedge$ a = $\sum_{e\in x'} e \,/\, \sum_{e\in x'} 1$)

Selection
t:enor(E) ~ x:infile($\mathbb{R}$) (st,e,x:read)
cond(e) ~ e≤0 $\vee$ st=abnorm
Two summations
f(e) ~ e, 1
s ~ s, c
H, +, 0 ~ ($\mathbb{R}$ +, 0.0), ($\mathbb{N}$ +, 0)

Algorithm:

st,e,x:read	e:$\mathbb{R}$
loop e>0 $\wedge$ st=norm	st:Status
st,e,x:read	
s, c := 0.0, 0 st,e,x:read	s:$\mathbb{R}$, c:$\mathbb{N}$
loop st=norm	
s, c := s+e, c+1	
st,e,x:read	
a := s / c	

c) Calculate the average of daily temperatures before and after the first temperature under the freezing point (in a sequential input file) if in the after version that first freezing temperature is included, too.

Specification:

$A = (\ x\text{:infile}(\mathbb{R}),\ a1,\ a2\text{:}\mathbb{R}\ )$

$Pre = (\ x = x_0 \wedge |x| \geq 2 \wedge \exists i \in [2..|x|]: x_0[i] \leq 0)$

$Post = (\quad (a1, (st', e', x')) = \sum_{e \in x_0}^{e \geq 0} e \,/\, \sum_{e \in x_0}^{e \geq 0} 1$

$\quad\wedge\ a2 = \sum_{e \in (e', x')} e \,/\, \sum_{e \in (e', x')} 1\ )$

Two summations

t:enor(E) ~ x:infile(ℝ) (st,e,x:read)
as long as e≥0

f(e) ~ e, 1

s ~ s1, c1

(H, +, 0) ~ (ℝ +, 0.0), (ℕ +, 0)

Two summations

t:enor(E) ~ x:infile(ℝ) (st,e,x:read)
next() instead of first()

f(e) ~ e, 1

s ~ s2, c2

(H, +, 0) ~ (ℝ +, 0.0), (ℕ +, 0)

Algorithm:

s1, c1 := 0.0, 0
st,e,x:read

e≥0 ∧ st=norm

s1, c1 := s1+e1, c1+1
st,e,x:read

a1 := s1 / c1

s2, c2 := 0.0, 0
st,e,x:read

st=norm

s2, c2 := s2+e, c2+1
st,e,x:read

a2 := s2 / c2

e, s1:ℝ, c1:ℕ
st:Status

s2:ℝ, c2:ℕ

4. In a sequential input file, elevation measurements are stored. A measurement is considered as a valley if the measurements before and after are higher than the actual one. How high is the highest valley?

Specification:

$A = (\ x\text{:infile}(\mathbb{R}),\ l\text{:}\mathbb{L},\ max\text{:}\mathbb{R}\ )$

$Pre = (\ x = x_0)$

$A = (\ t\text{:enor}(\mathbb{R} \times \mathbb{R} \times \mathbb{R}),\ l\text{:}\mathbb{L},\ max\text{:}\mathbb{R}\ )$

$Pre = (\ t = t_0)$

$Post = (\ (l, max) = \mathbf{MAX}_{\substack{(e,a,k) \in t_0 \\ e > a < k}}\ a\ )$

Cond. max. search.

t:enor(E) ~ t:enor(ℝ×ℝ×ℝ),
where (e, a, k) := t.current()

f(e) ~ a

cond(e) ~ e>a<k

H, < ~ ℤ, <

Algorithm:

l := false
t.first()

¬t.end()

(e, a, k) := t.current()

¬(e>a<k)	e>a<k ∧ l		e>a<k ∧ ¬l
—	max < a		l, max := true, a
	max := a	—	

t.next()

e, a, k: ℝ

Enumerator

$(\mathbb{R}\times\mathbb{R}\times\mathbb{R})^*$	first()	next()	(e, a, k) := current() e, a, k: $\mathbb{R}$	l := end() l:$\mathbb{L}$
x:infile($\mathbb{R}$) e, a, k: $\mathbb{R}$ st : Status	st, e, x:read st, a, x:read st, k, x:read	e := a a := k st, k, x:read	(e, a, k)	l := st=abnorm