



EÖTVÖS LORÁND
UNIVERSITY | BUDAPEST


Custom Enumerators

Object Oriented Programming | 2024 Spring
Practice 6

By Tarlan Ahadli

Enumerators in C# : foreach


- Simplicity:** Cleaner syntax without manual index management.
- Abstraction:** Enables uniform code for all enumerable collections.

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light blue font on a dark background.

```
string[] names = { "Alice", "Bob", "Charlie" };  
  
foreach (string name in names)  
{  
    Console.WriteLine(name);  
}
```

Enumerators in C# : while

- Simplicity:** Cleaner syntax without manual index management.
- Abstraction:** Enables uniform code for all enumerable collections.



```
string[] names = { "Alice", "Bob", "Charlie" };  
IEnumerator enumerator = names.GetEnumerator();  
  
while (enumerator.MoveNext())  
{  
    string name = (string)enumerator.Current;  
    Console.WriteLine(name);  
}
```

Problem No. 1

The integers in the input file are given in ascending order. Count the frequency of each number in the file and save in sequential output file.

$infile(Z) \rightarrow \langle 2, 2, 3, 4, 4, 4 \rangle \rightarrow outfile(rec(Z, N)) \rightarrow \langle (2, 2), (3, 1), (4, 3) \rangle$

$A = (x: infile(Z), y: outfile(Stat))$

$Stat = rec(num: Z, count: N)$

$Pre = (x = x' \wedge x \uparrow)$

$Post = (\forall n \in \mathbb{Z}, \exists! f \in \mathbb{N} \mid (n, f) \in outfile \wedge f = count(n, infile))$

- \forall (For All)
- $\exists!$ (Exists Exactly One)
- \in (Element Of)
- \mathbb{N} (Natural Numbers)
- \mathbb{Z} (Integers)
- \wedge (And)

For every integer n that exists in our set of integers, there is a unique frequency f that is a natural number. In the output file, the pair (n, f) will be included, where f is the count of how many times n appears in the input file.

Problem: Solve with enumerator

$A = (x:infile(Z), y:outfile(Stat))$

$Stat = rec(num:Z, count:N)$

$Pre = (x = x' \wedge x \uparrow)$

$A = (t:enor(Stat), y:outfile(Stat))$

$Pre = (t = t')$

$Post = (y = t') = \underbrace{(y = \bigoplus_{e \in t'} \langle e \rangle)}_{\text{Summation pattern}}$

Analogy: Summation

$s \rightarrow y$

$f(i) \rightarrow e$

$E(\text{type}) \rightarrow Stat$

$(H, +, 0) \rightarrow (Set(Stat), \bigoplus, \langle \rangle)$

Algorithm		
$y := \langle \rangle$		
$t.first()$		
.... $\neg t.end()$		
<table border="1"><tr><td>$y:write(t.current())$</td></tr><tr><td>$t.next()$</td></tr></table>	$y:write(t.current())$	$t.next()$
$y:write(t.current())$		
$t.next()$		

Create Custom Enumerator: $t:enor(Stat)$

- **Value**
 - $enor(Stat)$
- **Operations**
 - $first()$
 - $next()$
 - $current()$
 - $end()$
- **Representation**
 - $x:infile(Z)$
 - $st:Status$
 - $e:Z$
 - $current: Stat$
 - $end: L$
- **Implementation**
 - $current()$
 - return current
 - $end()$
 - return true/false
 - $first()$
 - $st,e,x:read$
 - $next()$
 - $next()$
 - $A = (x:infile, st:Status, e:Z, current: Stat, end: L)$
 - $Pre = (x = x' \wedge st = st' \wedge x \uparrow)$
 - $Post = (end = (st' = abnorm) \wedge$

$$\neg end \Rightarrow (current.num = e' \wedge current.count \wedge (st, e, x) = \sum_{e \in (e', x')}^{e=current.num} 1))$$

Create Custom Enumerator: $t:enor(Stat)$

- Analogy
 - $t:enor(E) \rightarrow x:infile(Z) \mid (st, e, x)$ as long as $e = current.num$
 - $c \rightarrow current.count$
 - $cond(i) \rightarrow true \mid$ we don't have a condition

Algorithm	
end := (st = abnorm)	
$\neg end()$	
true	false
current.num = e	
current.counter = 0	
st = norm ^ e = current.num	
current.counter = current.counter + 1	
st,e,x:read()	

Problem No. 2

We have a sequential input file containing transactions from a bank. Each transaction contains an account number, customer name, date of the transaction, and amount. The transactions are sorted in ascending order by account number. Write the account numbers and their summed balances in pairs (Acc-Am) to a sequential output file, where the summed balance (Bal) for the account is less than -1000 euros.

$A = (x:infile(Transaction), y:outfile(Bal))$

$Transaction = rec(acc:S, date:S, amount:R)$

$Bal = rec(acc:S, amount:R)$

$Pre = (x = x' \wedge x \uparrow acc)$

$A = (t:enor(Bal), y:outfile(Bal))$

$Pre = (t = t')$

$Post = (y = \bigoplus_{\substack{e \in t' \\ e.amount < -1000}} \langle e \rangle)$

Analogy: Conditional Summation

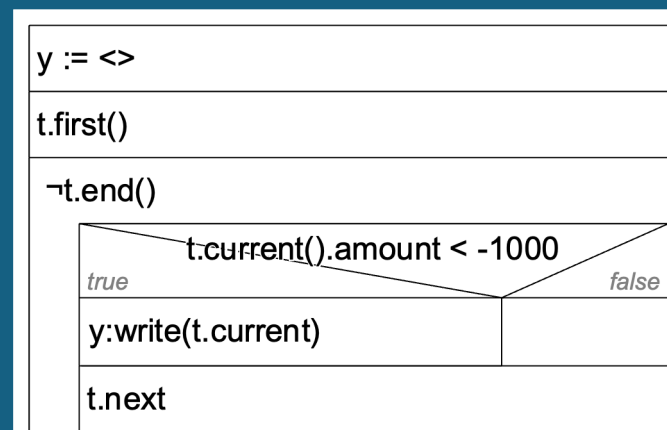
$s \rightarrow y$

$f(i) \rightarrow e$

$cond(e) \rightarrow e.amount < -1000$

$E(type) \rightarrow Bal$

$(H, +, 0) \rightarrow (Set(Bal), \oplus, \langle \rangle)$



Custom Enumerator: t:enor(Bal)

- **Value**
 - enor(Bal)
- **Operations**
 - first()
 - next()
 - current()
 - end()
- **Representation**
 - x:infile(Transaction)
 - st:Status
 - e:Trans
 - current: Bal
 - end: L
- **Implementation**
 - current()
 - return current
 - end()
 - return true/false
 - first()
 - st,e,x:read
 - next()
 - next()
 - $A = (x:infile(Trans), st:Status, e:Trans, current: Bal, end: L)$
 - $Pre = (x = x' \wedge st = st' \wedge e = e' \wedge x \uparrow acc)$
 - $Post = (end = (st' = abnorm) \wedge$

$$\neg end \Rightarrow (current.acc = e'.acc \wedge curr.am \wedge (st, e, x) = \sum_{e \in (e', x')}^{e.acc=curr.acc} e.am))$$

Create Custom Enumerator: $t:enor(Stat)$


- Analogy : Summation
 - $s \rightarrow curr.am$
 - $f(e) \rightarrow e.am$
 - $t:enor(E) \rightarrow x:infile(Trans) \mid (st, e, x)$ as long as $e.acc = current.acc$
 - $c \rightarrow current.count$
 - $cond(i) \rightarrow true \mid$ we don't have a condition
 - $(H, +, 0) \rightarrow (R, +, 0)$

end := (st = abnormal)	
$\neg end$	
true	false
curr.acc := e.acc	
curr.am := 0	
st = norm \wedge e.ac = curr.acc	
curr.am := curr.am + e.am	
st,e,x : read	

General advice

1. First assume you have enumerator : solve 1st part of problem
2. Then implement enumerator
 1. first()
 2. current()
 3. end()
 4. next()
3. For condition always check if enumeration ended

Sequential Vs Non-Sequential

Feature	Sequential File	Non-sequential File
Data Order	Records follow a sequence	Records can be in any order
Accessing Data	Read records one by one	Access any record directly
Example	Text file, log file	Database table, spreadsheet
 Export to Sheets		