Lazy	eval	luation	
------	------	---------	--

CSc 372	Haskell evaluates expressions using a technique called lazy evaluation:
Comparative Program Languages 14 : Haskell — Lazy Evalua	2. No shared expression is evaluated more than once; if the expression is ever evaluated then the result is shared between all those places in which it is used
Christian Collberg	Lazy functions are also called non-strict and evaluate their arguments lazily or by need.
collberg+372@gmail.com Department of Computer Science University of Arizona	C functions and Java methods are strict and evaluate their arguments eagerly.
Copyright ⓒ 2005 Christian Collberg	
—Fall 2005 — 14 [1]	372 — Fall 2005 — 14 [2]

Don't Evaluate Until Necessary

The first of these ideas is illustrated by the following function:

```
ignoreArgument x = "I didn't evaluate x"
```

Since the result of the function ignoreArgument doesn't depend on the value of its argument x, that argument will not be evaluated:

```
$ hugs +s
> ignoreArgument (1/0)
I didn't evaluate x
(246 reductions, 351 cells)
```

Don't Evaluate Until Necessary...

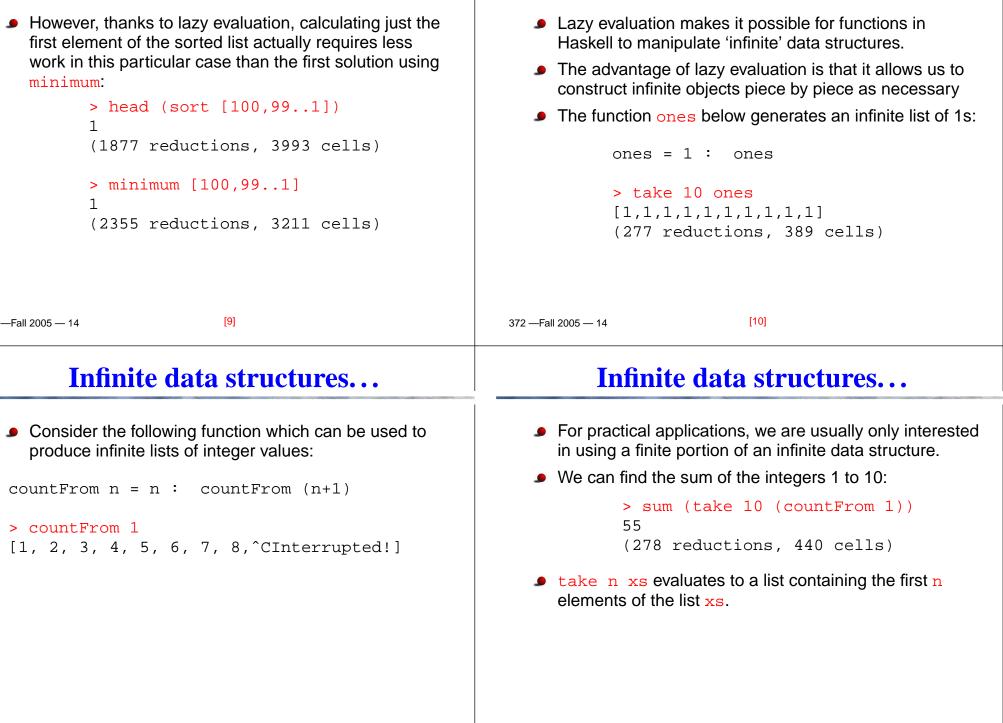
The function seq forces strict evaluation when that is necessary:

```
> seq ignoreArgument (1/0)
Inf
(32 reductions, 78 cells)
```

Evaluate Shared Expressions Once
 Notice that the first expression requires fewer reduction than the second. A reduction is the basic step of evaluating a Haskell expression, by applying a function to its argument.
372 — Fall 2005 — 14 [6]
Taking the Minimum
 Consider the task of finding the smallest element of a list of integers. minimum [100,991] (2355 reductions, 3211 cells) [100,991] denotes the list of integers from 1 to 100 arranged in decreasing order. Instead, we could first sort and then take the head of the result: :load List sort [100,991] 2, 3, 4, 5, 6, 7, 8,, 99, 100] (3430 reductions, 8234 cells)

Taking the Minimum...

Infinite data structures



Infinite data structures	Infinite data structures
 Infinite data structures enable us to describe an object without being tied to one particular application of that object. The following definitions for infinite list of powers of two [1, 2, 4, 8,]: powersOfTwo = 1 : map double powersOfTwo where double n = 2*n take 10 powersOfTwo [1, 2, 4, 8, 16, 32, 64, 128, 256, 512] 	 xs!!n evaluates to the n:th element of the list xs. We can define a function to find the nth power of 2 for any given integer n: powersOfTwo = 1 : map (*2) powersOfTwo twoToThe n = powersOfTwo !! n twoToThe 5 32
—Fall 2005 — 14 [13]	372 — Fall 2005 — 14 [14]
Fibonacci	Acknowledgements
 Here's a definition of a function that generates an infinite list of all the fibonacci numbers: <pre>fib = 1:1:[a+b a,b <-zip fib (tail fib)]</pre> take 10 fib <pre>[1,1,2,3,5,8,13,21,34,55]</pre> 	 These slides were derived mostly from the Gofer manual. Functional programming environment, Version 2.20 © Copyright Mark P. Jones 1991.