CSE 3302
Programming Languages

Functional Programming Language: Haskell (cont'd)

Chengkai Li
Spring 2008

Grading

- Homework (HW): 25%. (HW1, HW2, HW3, HW4, HW5)
- Machine Problems (MP): 20%. (MP1, MP2)
- Essays (ES): 10%.
- Midterm exam: 20%.
- Final exam: 25%.
- Bonus points: 5%. Based on class participation.
  some additional bonus points (in MP2, HW5)

Letter Grades

- curve-based
- The cutoffs for letter grades are based on your performance.
- Bonus point: can only increase your grade
  Example:
  - cutoff for A: 88.5
  - your raw score: 86
  - bonus points: 3
  - your grade: B -> A (86+3>88.5)

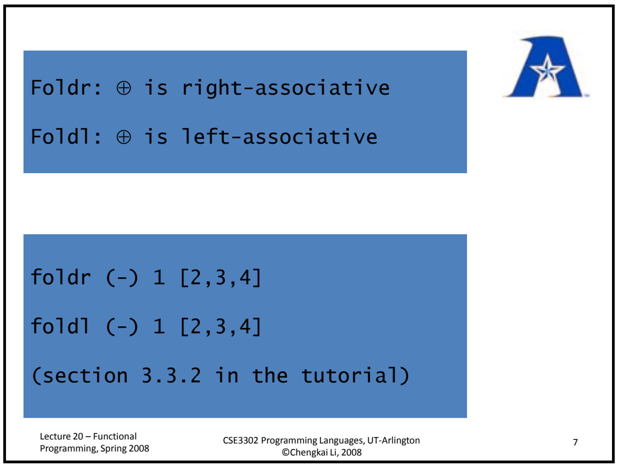
What's ahead

- HW4: due by April 21st
- HW5: due by May 2nd
- Essay: have you started yet? (due at May 1st)

Grades on WebCT

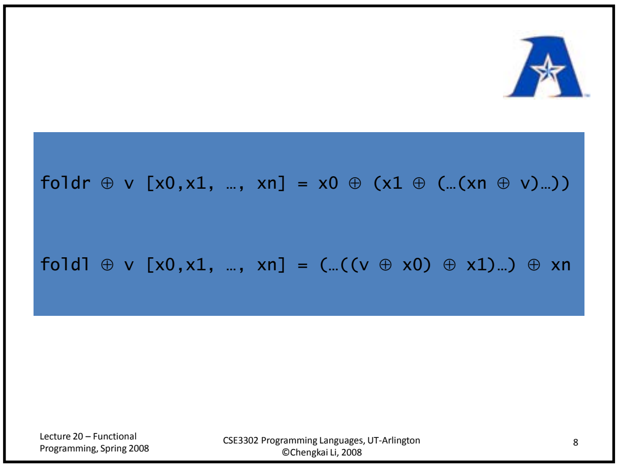
Foldl and Foldr

```
foldl            :: (a -> b -> a) -> a -> [b] -> a
foldl f z []       = z
foldl f z (x:xs)   = foldl f (f z x) xs


foldr            :: (a -> b -> b) -> b -> [a] -> b
foldr f z []       = z
foldr f z (x:xs)   = f x (foldr f z xs)
```

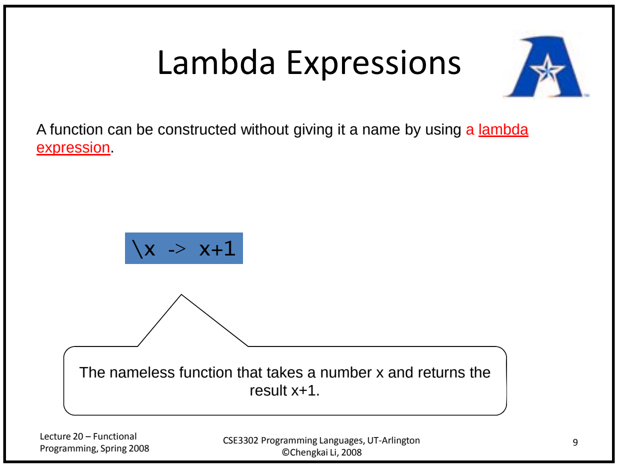


```
Foldr: ⊕ is right-associative

Foldl: ⊕ is left-associative
```

```
foldr (-) 1 [2,3,4]

foldl (-) 1 [2,3,4]

(section 3.3.2 in the tutorial)
```



```
foldr ⊕ v [x0,x1, …, xn] = x0 ⊕ (x1 ⊕ (…(xn ⊕ v)…))

foldl ⊕ v [x0,x1, …, xn] = (…((v ⊕ x0) ⊕ x1)…) ⊕ xn
```

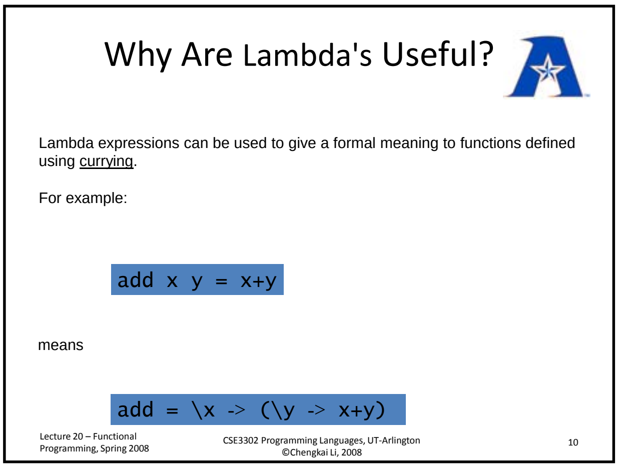


Lambda Expressions

A function can be constructed without giving it a name by using a lambda expression.

```
\x -> x+1
```

The nameless function that takes a number x and returns the result x+1.



Why Are Lambda's Useful?

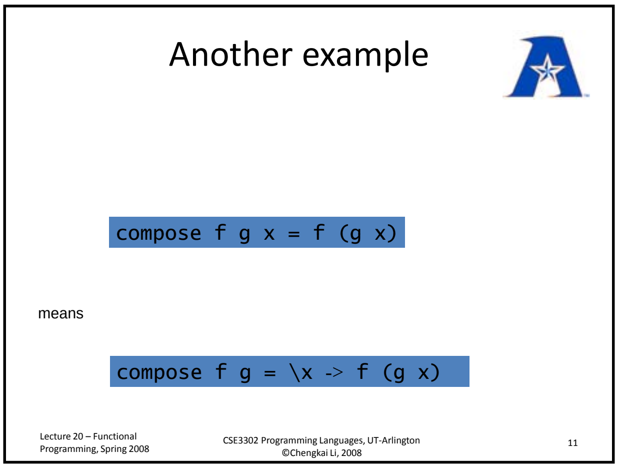
Lambda expressions can be used to give a formal meaning to functions defined using currying.

For example:

```
add x y = x+y
```

means

```
add = \x -> (\y -> x+y)
```

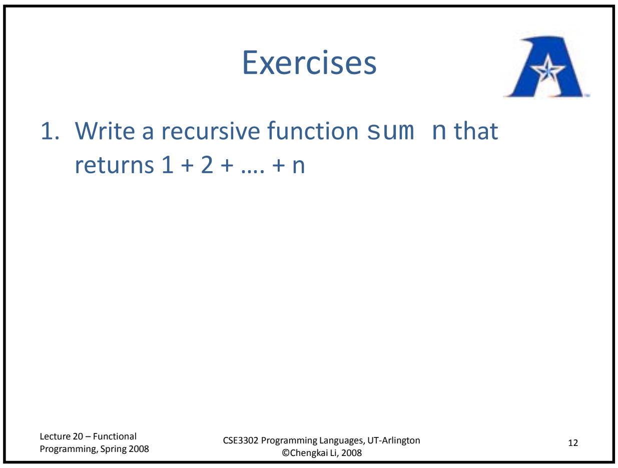


Another example

```
compose f g x = f (g x)
```

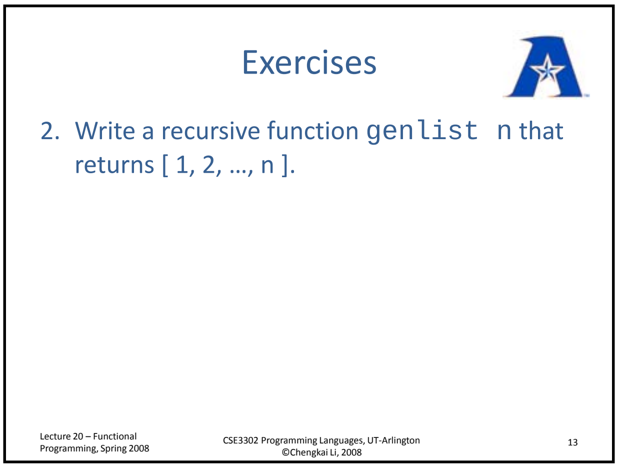
means

```
compose f g = \x -> f (g x)
```



Exercises

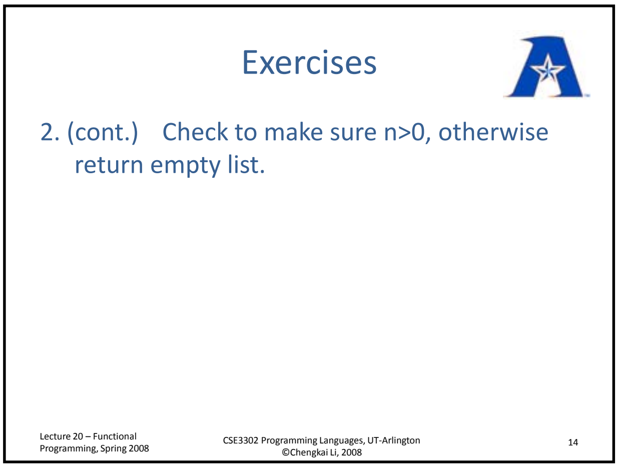
1. Write a recursive function sum n that returns 1 + 2 + …. + n



Exercises

2. Write a recursive function `genlist n` that returns [1, 2, …, n].

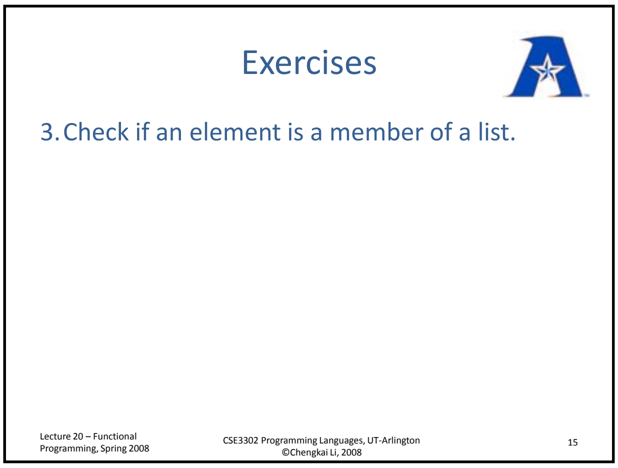
Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
13



Exercises

2. (cont.) Check to make sure n>0, otherwise return empty list.

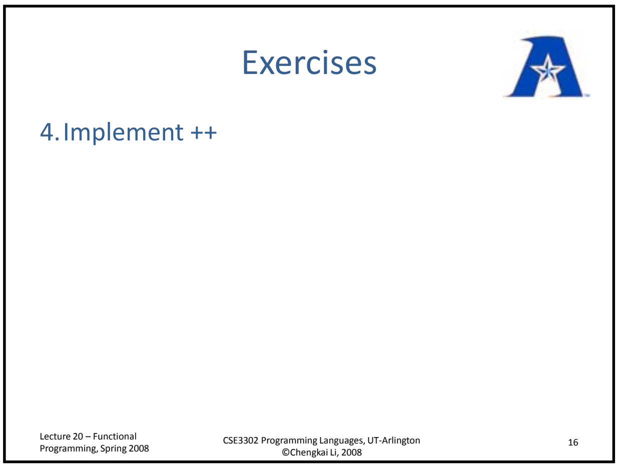
Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
14



Exercises

3. Check if an element is a member of a list.

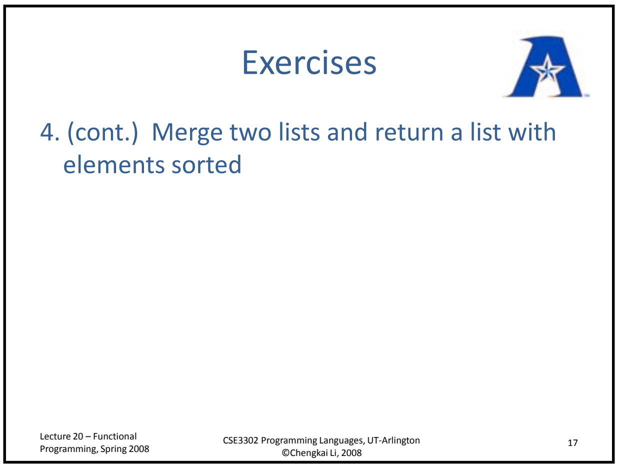
Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
15



Exercises

4. Implement ++

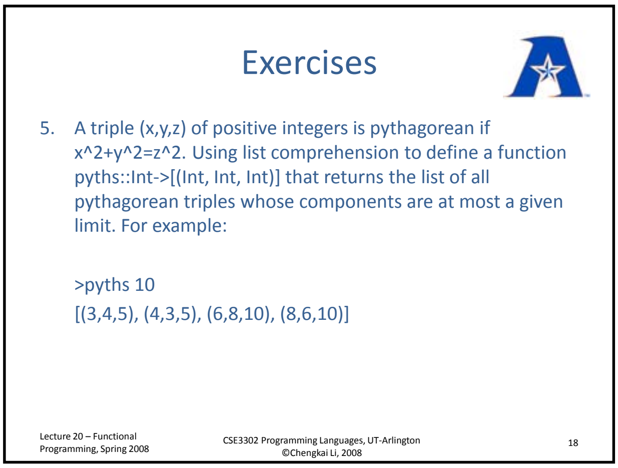
Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
16



Exercises

4. (cont.) Merge two lists and return a list with elements sorted

Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
17

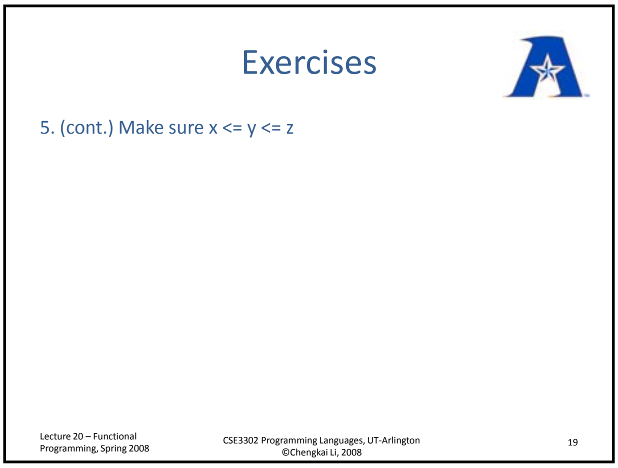


Exercises

5. A triple (x,y,z) of positive integers is pythagorean if x^2+y^2=z^2. Using list comprehension to define a function pyths::Int->[(Int, Int, Int)] that returns the list of all pythagorean triples whose components are at most a given limit. For example:

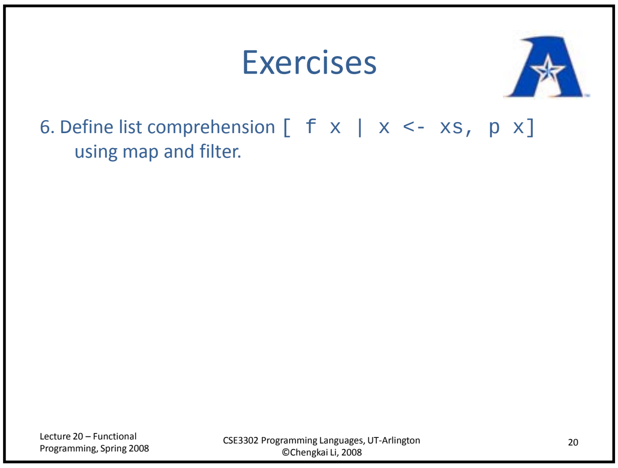
```
>pyths 10
[(3,4,5), (4,3,5), (6,8,10), (8,6,10)]
```

Lecture 20 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington
©Chengkai Li, 2008
18



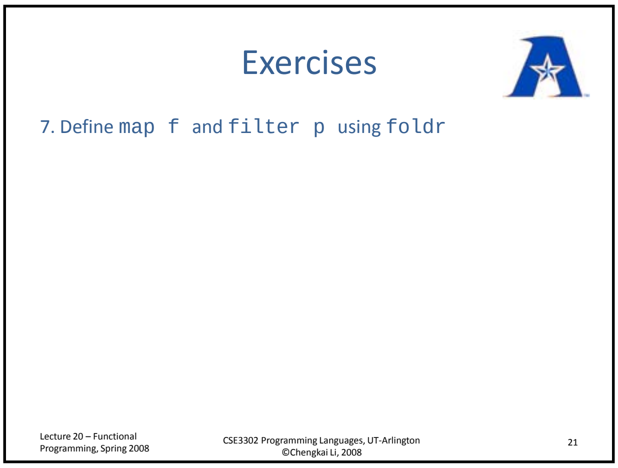
Exercises

5. (cont.) Make sure x <= y <= z



Exercises

6. Define list comprehension `[ f x | x <- xs, p x]` using map and filter.



Exercises

7. Define `map f` and `filter p` using `foldr`