CSE 3302 Programming Languages

Functional Programming Language: Haskell

Chengkai Li
Fall 2007

Reading and Implementation

- Yet Another Haskell Tutorial
  http://www.cs.utah.edu/~hal/htut
- WinHugs:
  http://cvs.haskell.org/Hugs/pages/downloading.htm
  Download file WinHugs-Sep2006.exe (14 MB):
- Installation:
  - Try to install outside of “Program Files”.
  - May fail if you install inside.

Topics

- Basics
- Types and classes
- Defining functions
- List comprehensions
- Recursive functions
- Higher-order functions

Notes

- You need to use script (module) files to define functions, and use interactive environment to evaluate expressions (functions)
- Haskell is not free-format. It has certain layout rules.
- General rules to follow when writing script files:
  - Indent the same amount for definitions at the same level
  - Don’t use tab

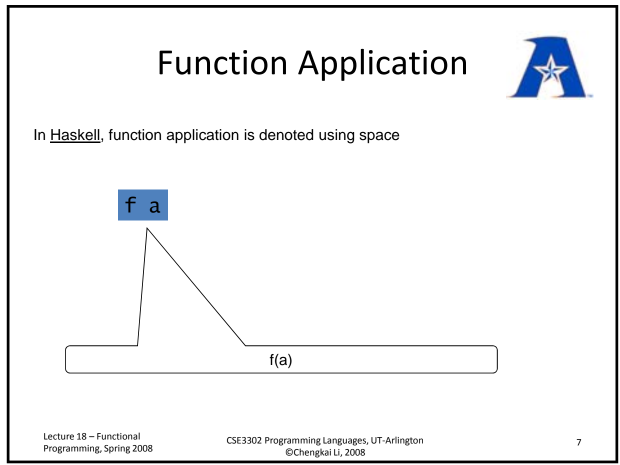
Basics

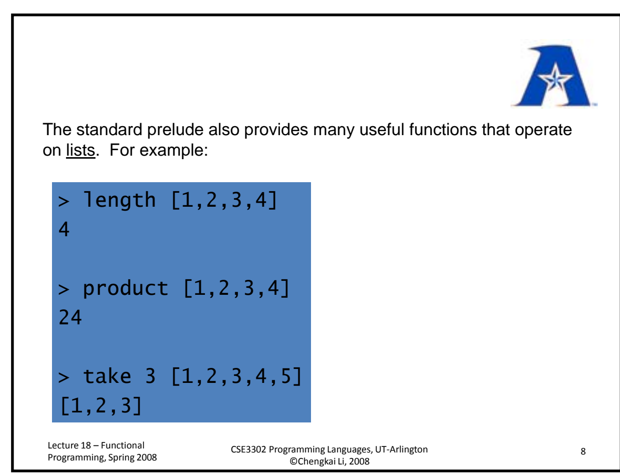
The Standard Prelude

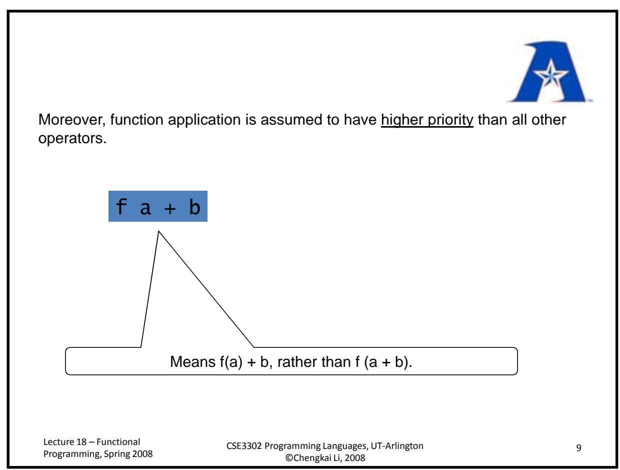
When Hugs is started it first loads the library file Prelude.hs, and then repeatedly prompts the user for an expression to be evaluated.

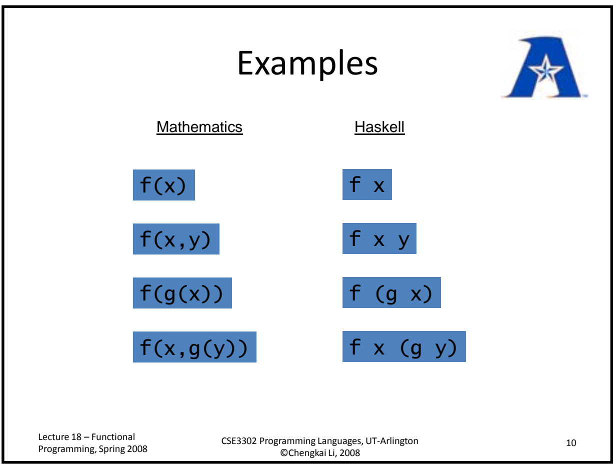
For example:

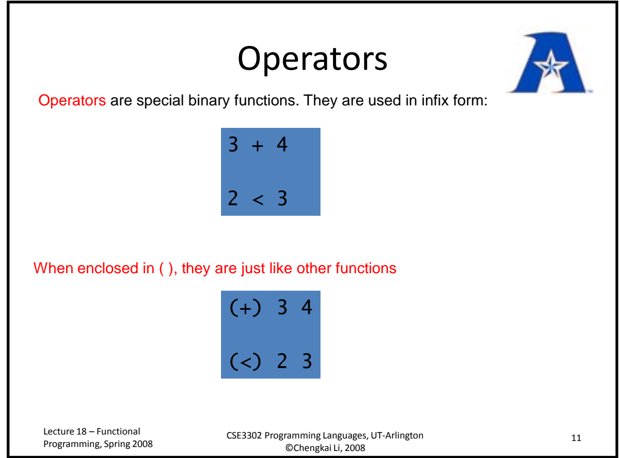
```
> 2+3*4
14

> (2+3)*4
20
```

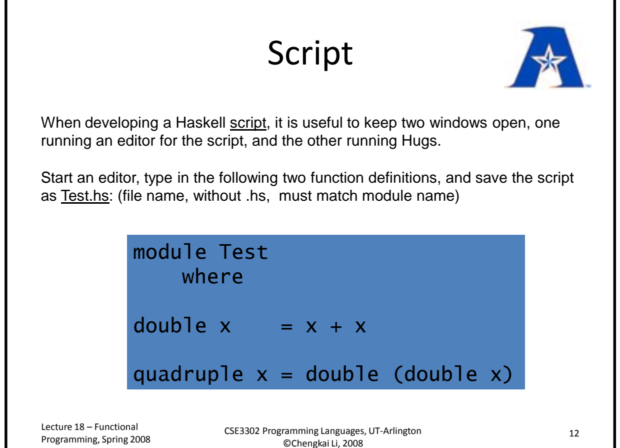

Function Application
In Haskell, function application is denoted using space
f a
f(a)
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
7

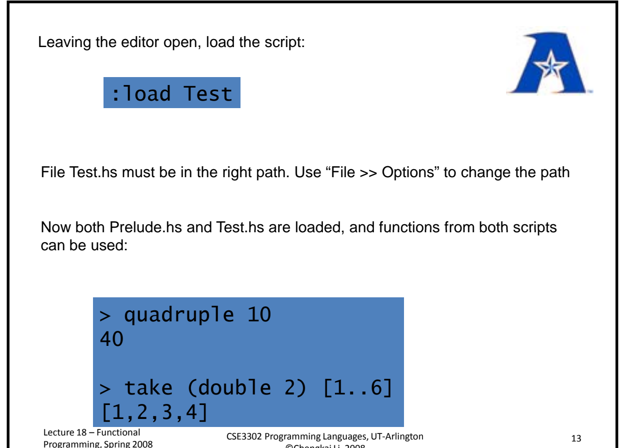

The standard prelude also provides many useful functions that operate on lists. For example:
> length [1,2,3,4]
4
> product [1,2,3,4]
24
> take 3 [1,2,3,4,5]
[1,2,3]
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
8

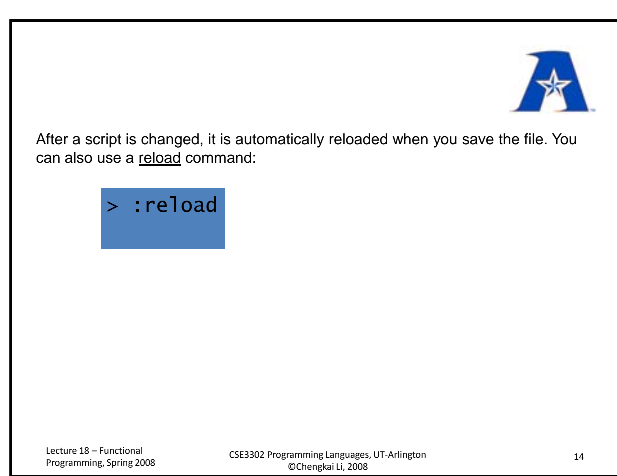

Moreover, function application is assumed to have higher priority than all other operators.
f a + b
Means f(a) + b, rather than f (a + b).
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
9

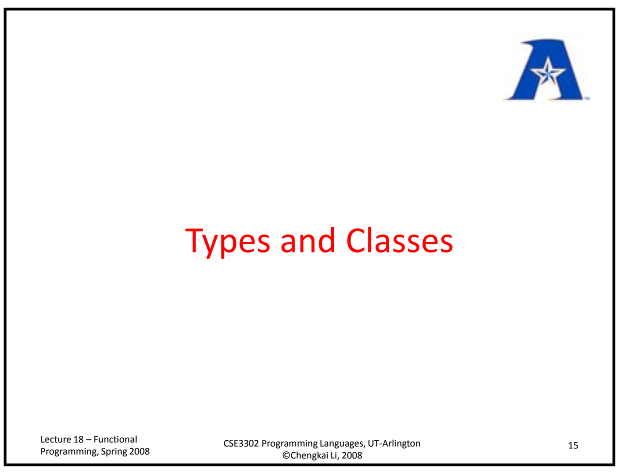

Examples
Mathematics
Haskell
f(x)
f x
f(x,y)
f x y
f(g(x))
f (g x)
f(x,g(y))
f x (g y)
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
10

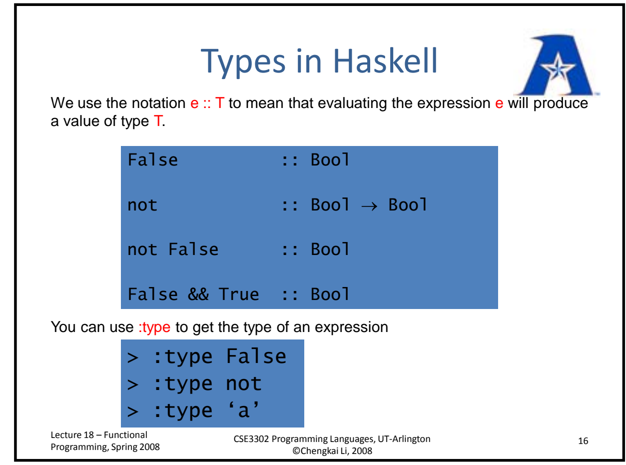

Operators
Operators are special binary functions. They are used in infix form:
3 + 4
2 < 3
When enclosed in ( ), they are just like other functions
(+) 3 4
(<) 2 3
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
11

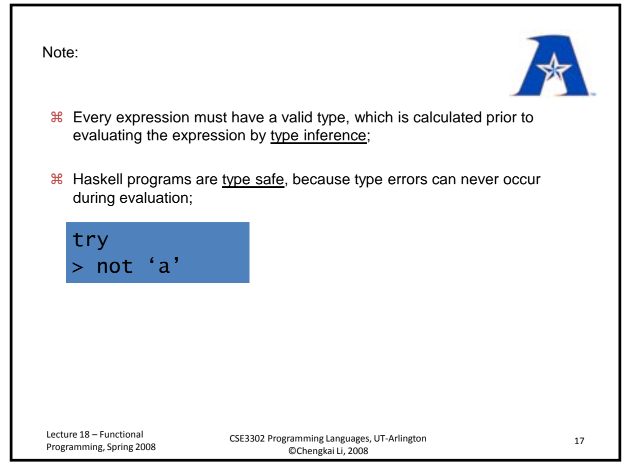

Script
When developing a Haskell script, it is useful to keep two windows open, one running an editor for the script, and the other running Hugs.
Start an editor, type in the following two function definitions, and save the script as Test.hs: (file name, without .hs, must match module name)
module Test
where
double x = x + x
quadruple x = double (double x)
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
12

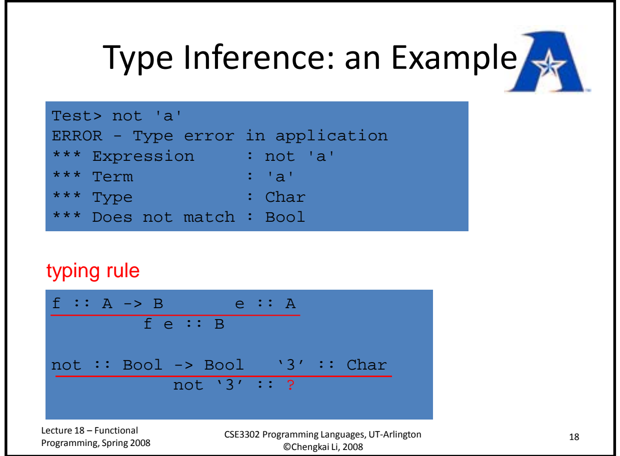

Leaving the editor open, load the script:
:load Test
File Test.hs must be in the right path. Use "File >> Options" to change the path
Now both Prelude.hs and Test.hs are loaded, and functions from both scripts can be used:
> quadruple 10
40
> take (double 2) [1..6]
[1,2,3,4]
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
13

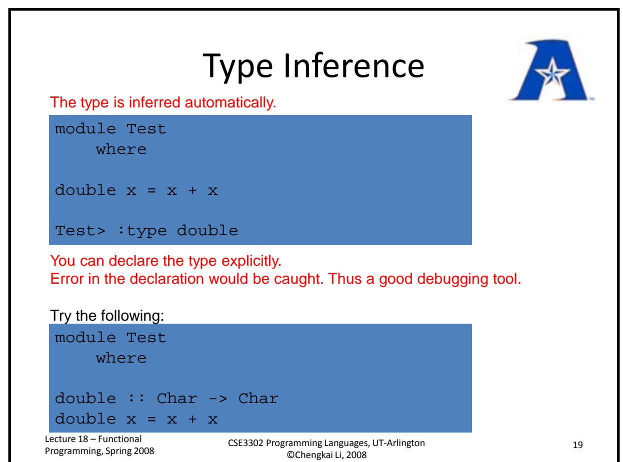

After a script is changed, it is automatically reloaded when you save the file. You can also use a reload command:
> :reload
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
14

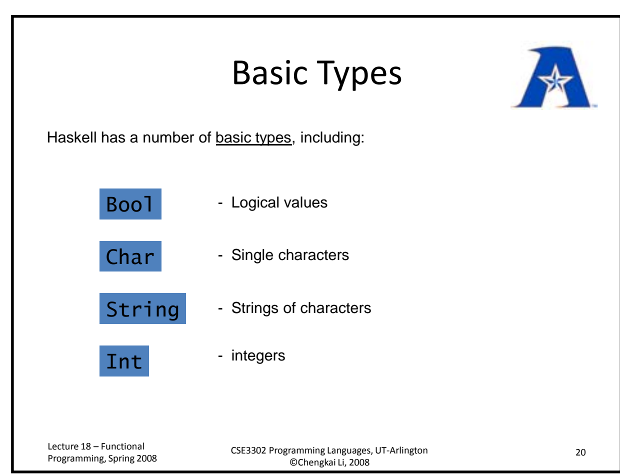

Types and Classes
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
15

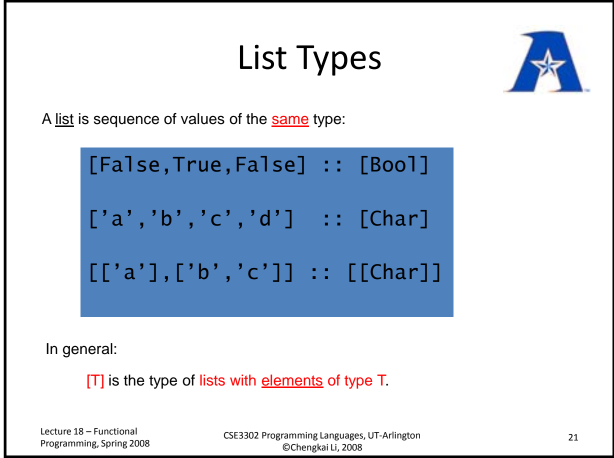

Types in Haskell
We use the notation e :: T to mean that evaluating the expression e will produce a value of type T.
False :: Bool
not :: Bool → Bool
not False :: Bool
False && True :: Bool
You can use :type to get the type of an expression
> :type False
> :type not
> :type 'a'
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
16

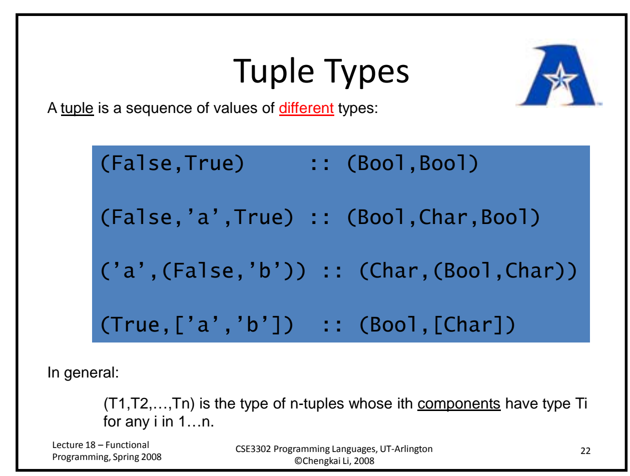

Note:
Every expression must have a valid type, which is calculated prior to evaluating the expression by type inference;
Haskell programs are type safe, because type errors can never occur during evaluation;
try
> not 'a'
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
17

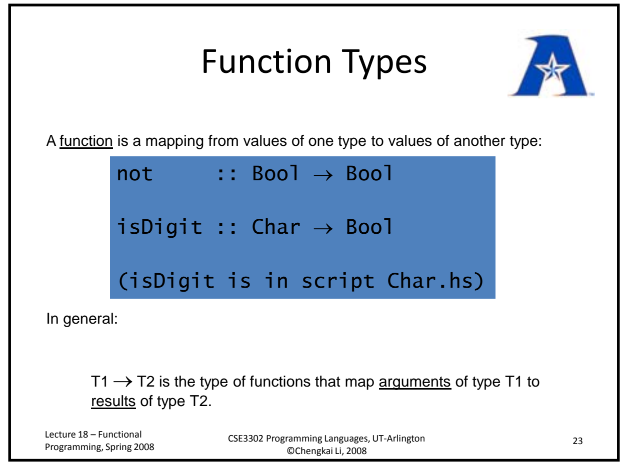

Type Inference: an Example
Test> not 'a'
ERROR - Type error in application
*** Expression : not 'a'
*** Term : 'a'
*** Type : Char
*** Does not match : Bool
typing rule
f :: A -> B e :: A
f e :: B
not :: Bool -> Bool '3' :: Char
not '3' :: ?
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
18

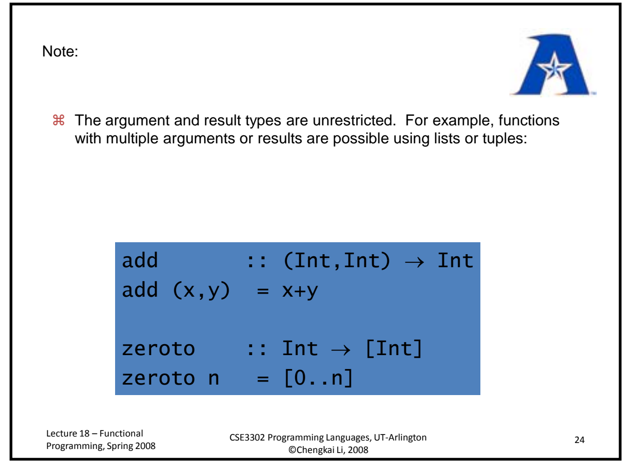

Type Inference
The type is inferred automatically.
module Test
where
double x = x + x
Test> :type double
You can declare the type explicitly.
Error in the declaration would be caught. Thus a good debugging tool.
Try the following:
module Test
where
double :: Char -> Char
double x = x + x
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
19

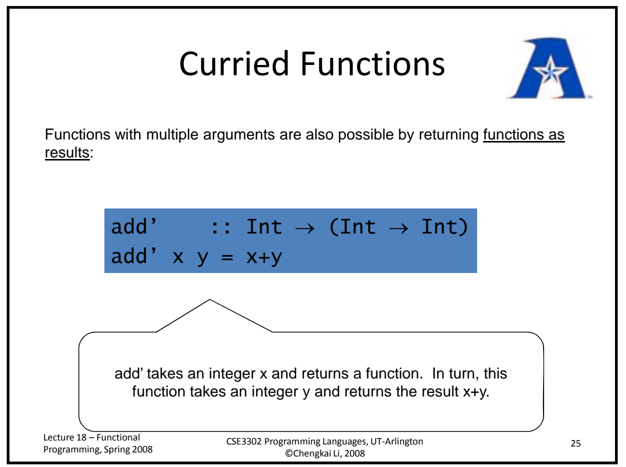

Basic Types
Haskell has a number of basic types, including:
Bool
- Logical values
Char
- Single characters
String
- Strings of characters
Int
- integers
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
20

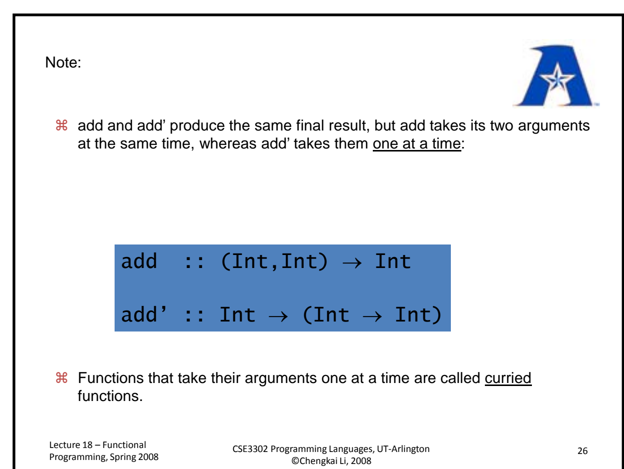

List Types
A list is sequence of values of the same type:
[False,True,False] :: [Bool]
['a','b','c','d'] :: [Char]
[['a'],['b','c']] :: [[Char]]
In general:
[T] is the type of lists with elements of type T.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
21

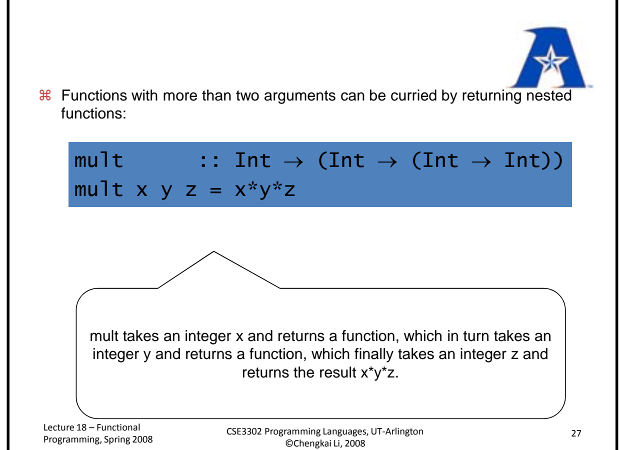

Tuple Types
A tuple is a sequence of values of different types:
(False,True) :: (Bool,Bool)
(False,'a',True) :: (Bool,Char,Bool)
('a',(False,'b')) :: (Char,(Bool,Char))
(True,['a','b']) :: (Bool,[Char])
In general:
(T1,T2,…,Tn) is the type of n-tuples whose ith components have type Ti for any i in 1…n.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
22

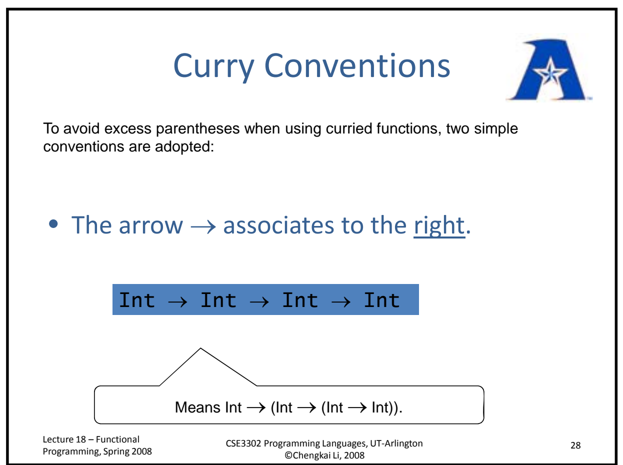

Function Types
A function is a mapping from values of one type to values of another type:
not :: Bool → Bool
isDigit :: Char → Bool
(isDigit is in script Char.hs)
In general:
T1 → T2 is the type of functions that map arguments of type T1 to results of type T2.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
23

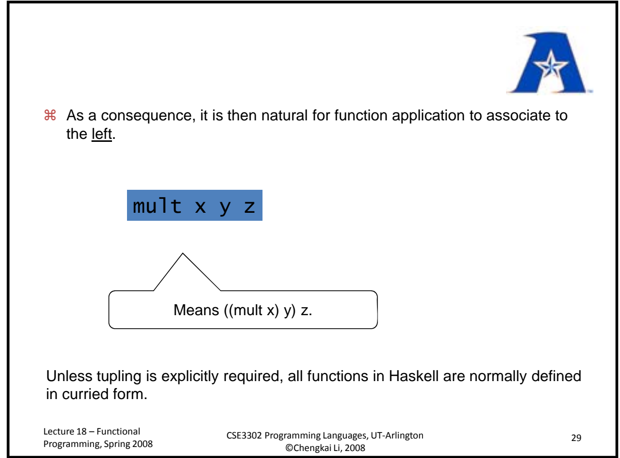

Note:
The argument and result types are unrestricted. For example, functions with multiple arguments or results are possible using lists or tuples:
add :: (Int,Int) → Int
add (x,y) = x+y
zeroto :: Int → [Int]
zeroto n = [0..n]
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
24


Curried Functions
Functions with multiple arguments are also possible by returning functions as results:
add' :: Int → (Int → Int)
add' x y = x+y
add' takes an integer x and returns a function. In turn, this function takes an integer y and returns the result x+y.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
25

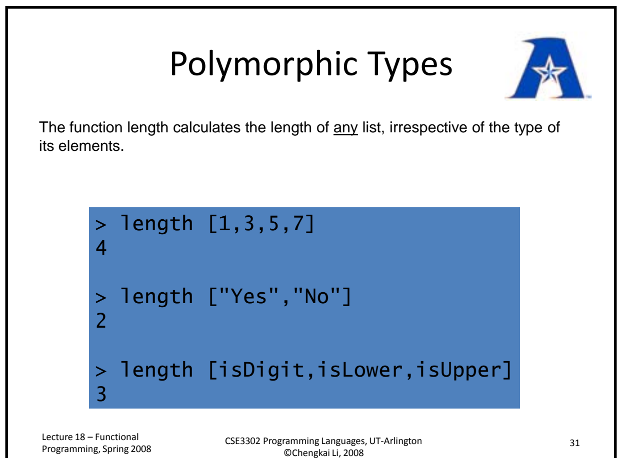

Note:
add and add' produce the same final result, but add takes its two arguments at the same time, whereas add' takes them one at a time:
add :: (Int,Int) → Int
add' :: Int → (Int → Int)
Functions that take their arguments one at a time are called curried functions.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
26

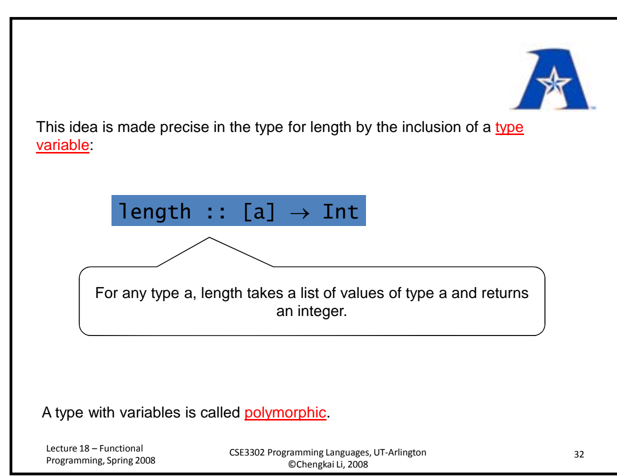

Functions with more than two arguments can be curried by returning nested functions:
mult :: Int → (Int → (Int → Int))
mult x y z = x*y*z
mult takes an integer x and returns a function, which in turn takes an integer y and returns a function, which finally takes an integer z and returns the result x*y*z.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
27

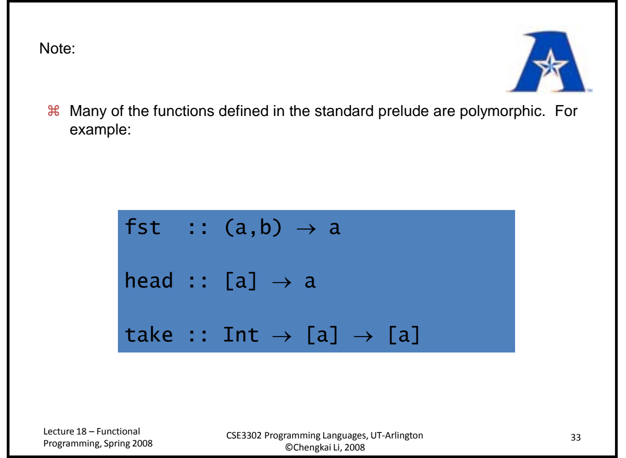

Curry Conventions
To avoid excess parentheses when using curried functions, two simple conventions are adopted:
The arrow → associates to the right.
Int → Int → Int → Int
Means Int → (Int → (Int → Int)).
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
28

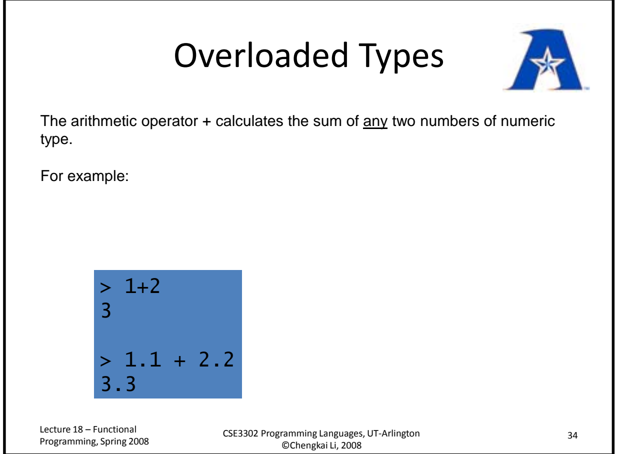

As a consequence, it is then natural for function application to associate to the left.
mult x y z
Means ((mult x) y) z.
Unless tupling is explicitly required, all functions in Haskell are normally defined in curried form.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
29

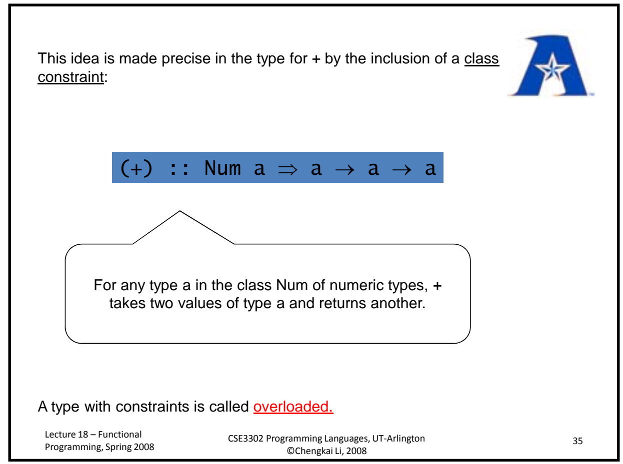
What's the big deal?
Why Currying?
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
30

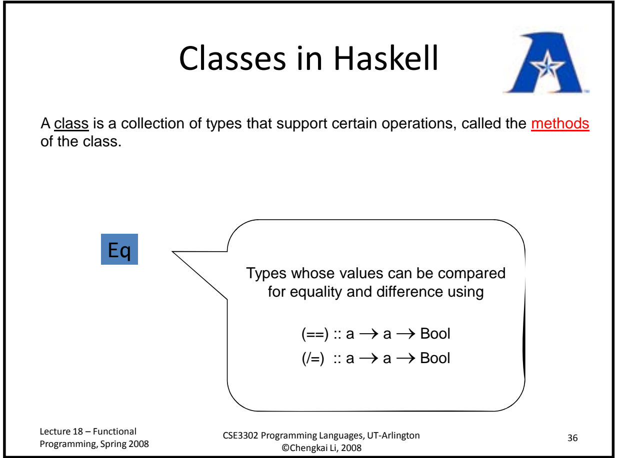

Polymorphic Types
The function length calculates the length of any list, irrespective of the type of its elements.
> length [1,3,5,7]
4
> length ["Yes","No"]
2
> length [isDigit,isLower,isUpper]
3
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
31

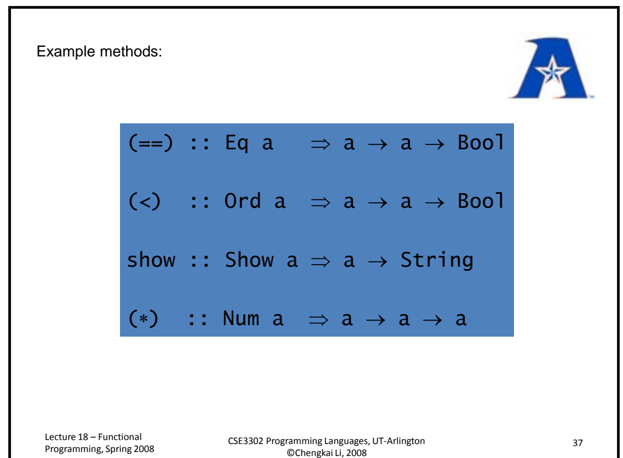

This idea is made precise in the type for length by the inclusion of a type variable:
length :: [a] → Int
For any type a, length takes a list of values of type a and returns an integer.
A type with variables is called polymorphic.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
32

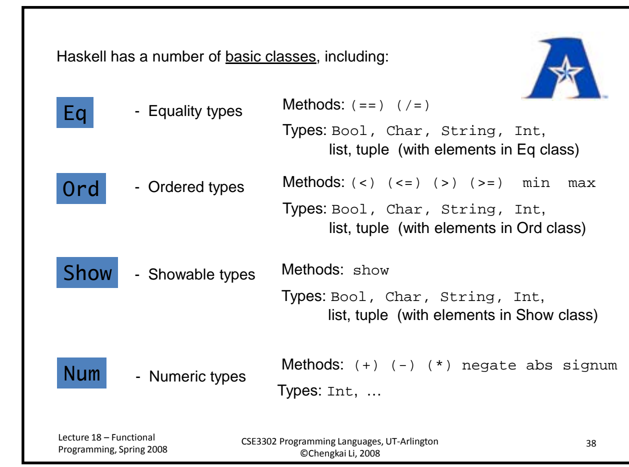

Note:
Many of the functions defined in the standard prelude are polymorphic. For example:
fst :: (a,b) → a
head :: [a] → a
take :: Int → [a] → [a]
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
33


Overloaded Types
The arithmetic operator + calculates the sum of any two numbers of numeric type.
For example:
> 1+2
3
> 1.1 + 2.2
3.3
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
34


This idea is made precise in the type for + by the inclusion of a class constraint:
(+) :: Num a ⇒ a → a → a
For any type a in the class Num of numeric types, + takes two values of type a and returns another.
A type with constraints is called overloaded.
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
35


Classes in Haskell
A class is a collection of types that support certain operations, called the methods of the class.
Eq
Types whose values can be compared for equality and difference using
(==) :: a → a → Bool
(/=) :: a → a → Bool
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
36


Example methods:
(==) :: Eq a ⇒ a → a → Bool
(<) :: Ord a ⇒ a → a → Bool
show :: Show a ⇒ a → String
(*) :: Num a ⇒ a → a → a
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
37


Haskell has a number of basic classes, including:
Eq
- Equality types
Methods: (==) (/=)
Types: Bool, Char, String, Int, list, tuple (with elements in Eq class)
Ord
- Ordered types
Methods: (<) (<=) (>) (>=) min max
Types: Bool, Char, String, Int, list, tuple (with elements in Ord class)
Show
- Showable types
Methods: show
Types: Bool, Char, String, Int, list, tuple (with elements in Show class)
Num
- Numeric types
Methods: (+) (-) (*) negate abs signum
Types: Int, …
Lecture 18 – Functional Programming, Spring 2008
CSE3302 Programming Languages, UT-Arlington ©Chengkai Li, 2008
38