

C/CPS 506

Comparative Programming Languages

Prof. Alex Ufkes









Topic 7: Types, type classes, custom types.

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

The copyright to this original work is held by Alex Ufkes. Students registered in course C/CPS 506 can use this material for the purposes of this course but no other use is permitted, and there can be no sale or transfer or use of the work for any other purpose without explicit permission of Alex Ufkes.

Course Administration (CCPS)

  CCPS506 - Comparative Programming La...      Alexander Ufkes 

Content Grades Assessment ▾ Communication ▾ Resources ▾ Classlist Course Admin

Haskell labs released today!

Any Questions?



Let's Get Started!

Types in Haskell

Statically Typed:

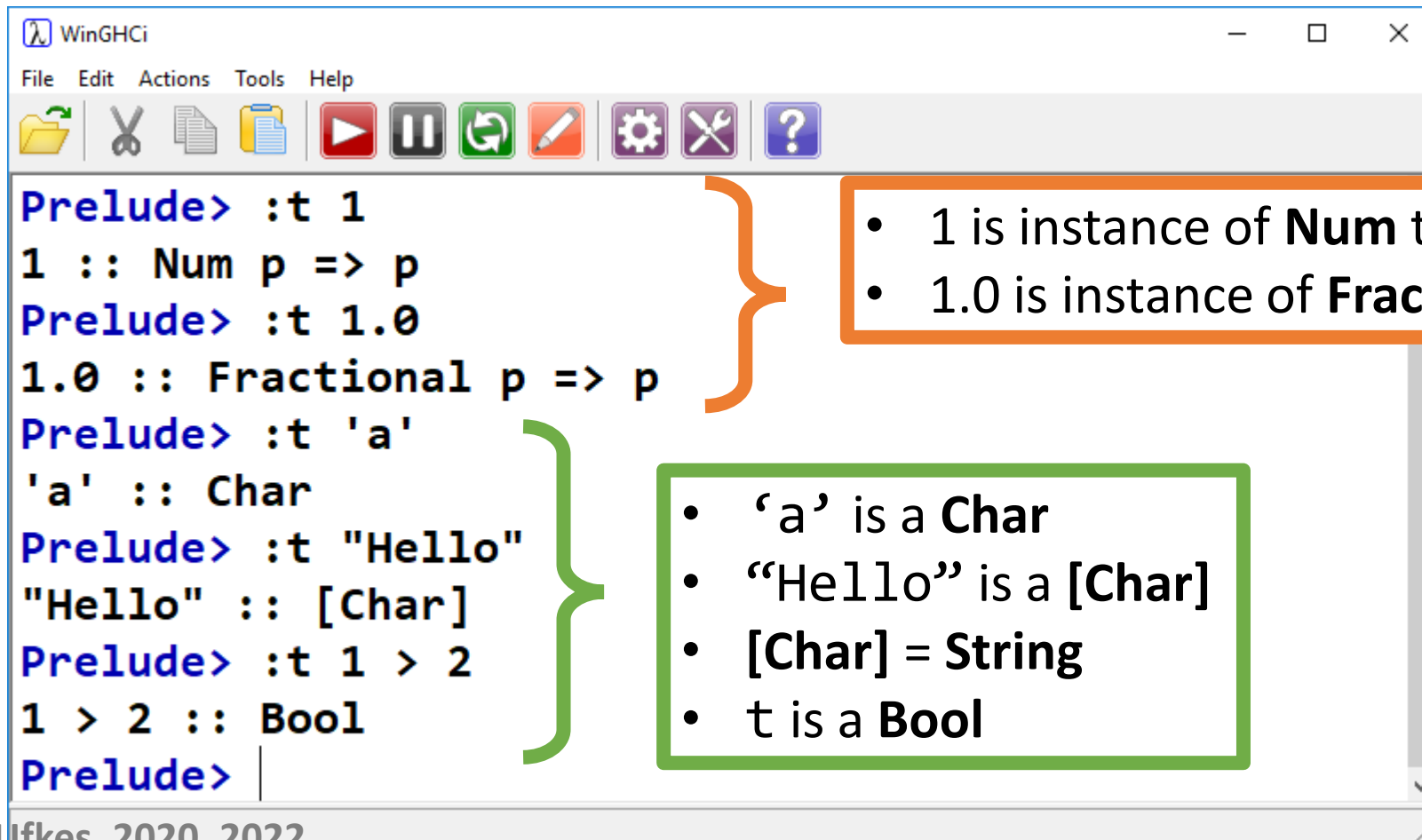
- Haskell uses static type checking.
- Every expression is assigned a type.
- If a function's arguments aren't the expected type, a compile error occurs.

Type Inference

- In Haskell, we need not specify type explicitly.
- It is inferred by the context: `X = "Hello"`, `X` is a string.
- However, we ***can*** explicitly specify types.
- Good practice when we know what types we want; compiler will give errors upon type mismatch.

Types in Haskell

`:t` can be used to reveal type:



The screenshot shows the WinGHCi window with the following content:

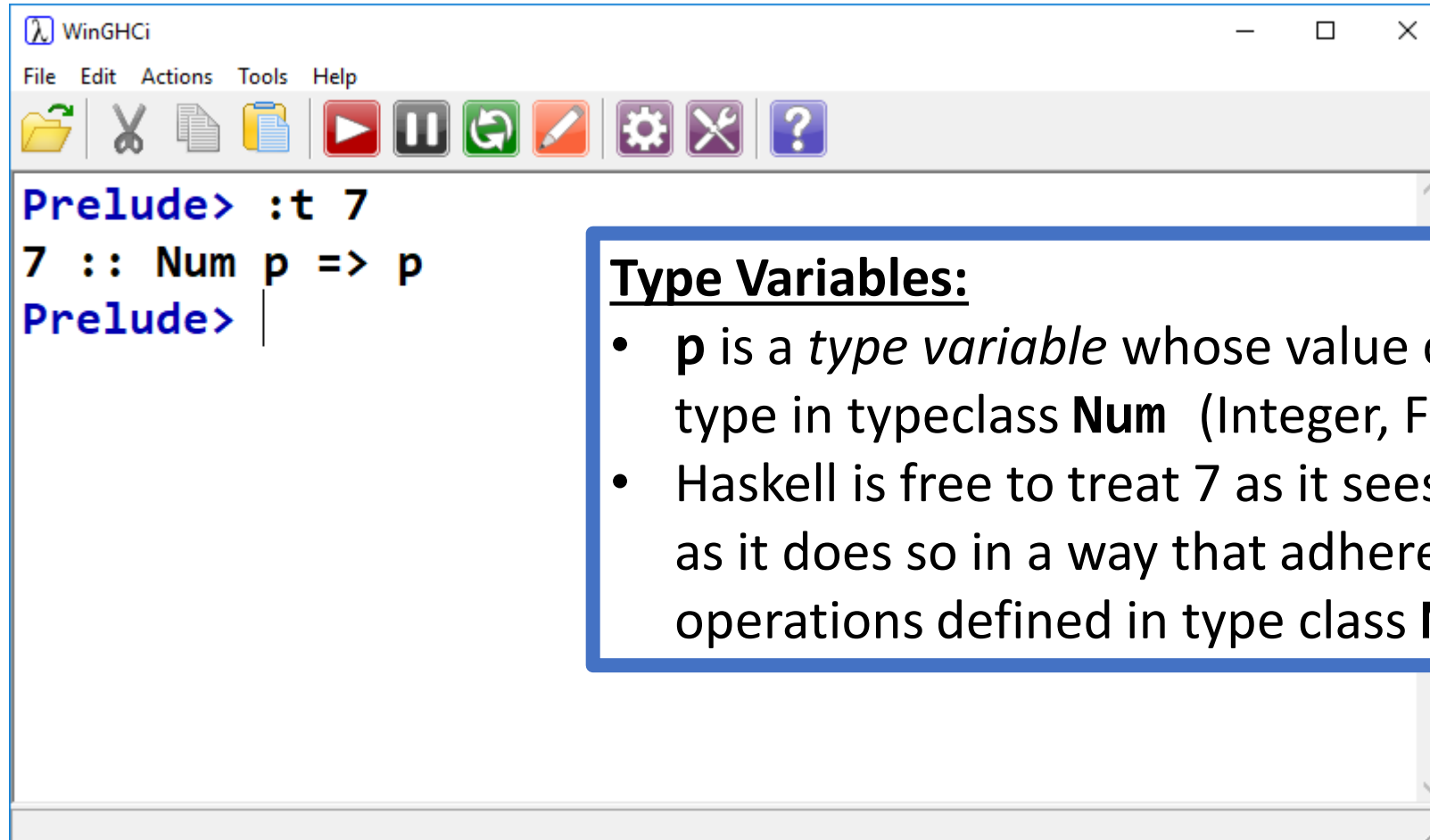
```
WinGHCi
File Edit Actions Tools Help
[Icons: Folder, Scissors, Document, Copy, Run, Pause, Refresh, Erase, Settings, Wrench, Help]

Prelude> :t 1
1 :: Num p => p
Prelude> :t 1.0
1.0 :: Fractional p => p
Prelude> :t 'a'
'a' :: Char
Prelude> :t "Hello"
"Hello" :: [Char]
Prelude> :t 1 > 2
1 > 2 :: Bool
Prelude>
```

Annotations in the image:

- An orange bracket groups the first two queries, pointing to an orange box with the following text:
 - 1 is instance of **Num** type class.
 - 1.0 is instance of **Fractional** type class.
- A green bracket groups the last three queries, pointing to a green box with the following text:
 - 'a' is a **Char**
 - "Hello" is a **[Char]**
 - **[Char]** = **String**
 - `t` is a **Bool**

Num p => p ?



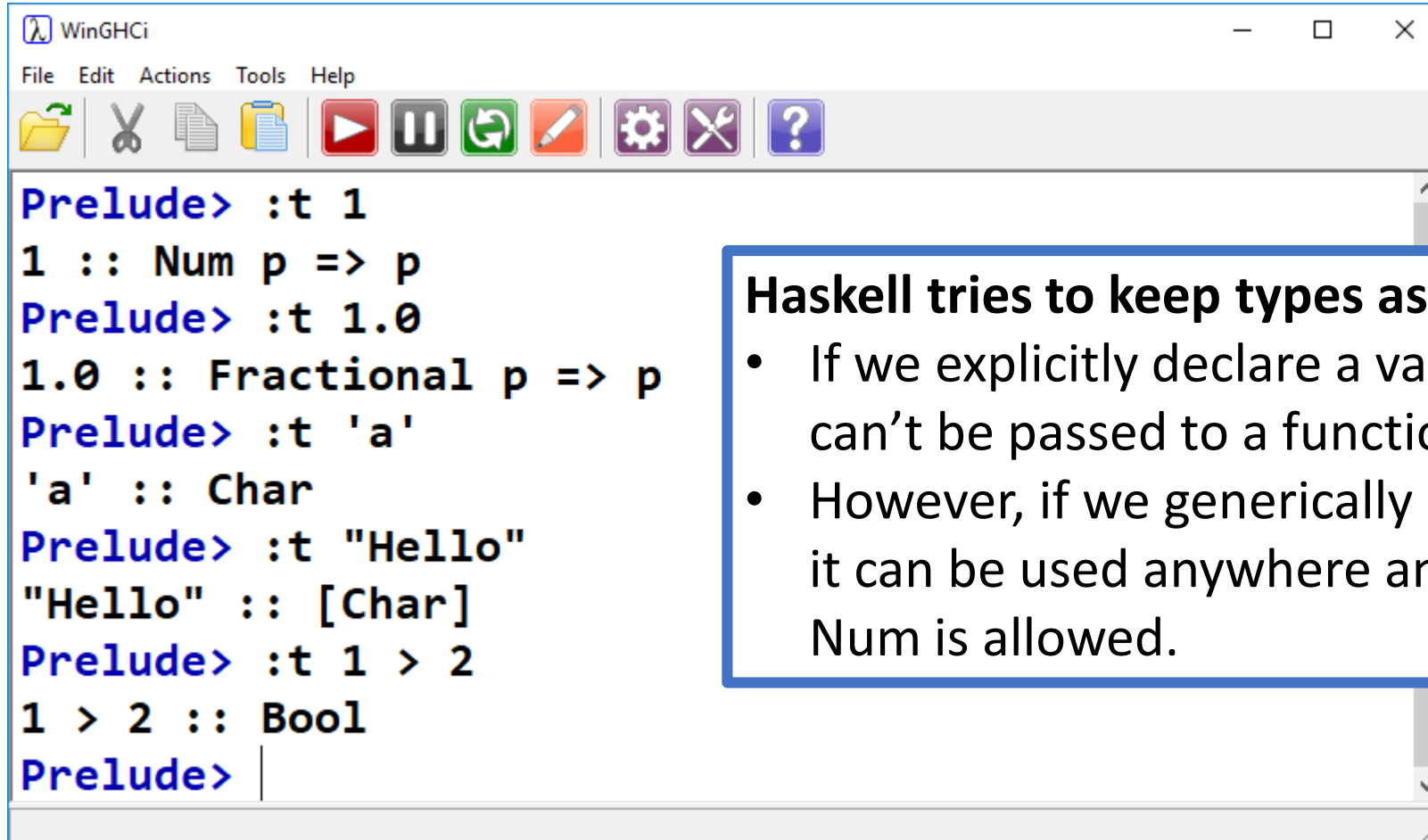
A screenshot of the WinGHCi Haskell interpreter window. The window has a title bar 'WinGHCi' and a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (folder, copy, paste), execution (play, pause, refresh), and settings (gear, wrench, question mark). The main text area shows the following interaction:

```
Prelude> :t 7
7 :: Num p => p
Prelude> |
```

Type Variables:

- **p** is a *type variable* whose value can be any type in typeclass **Num** (Integer, Float, etc.)
- Haskell is free to treat 7 as it sees fit, so long as it does so in a way that adheres to the operations defined in type class **Num**.

Typeclasses?



```
WinGHCi
File Edit Actions Tools Help
[Icons: Folder, Scissors, Document, Clipboard, Play, Pause, Refresh, Pencil, Gear, Wrench, Question Mark]

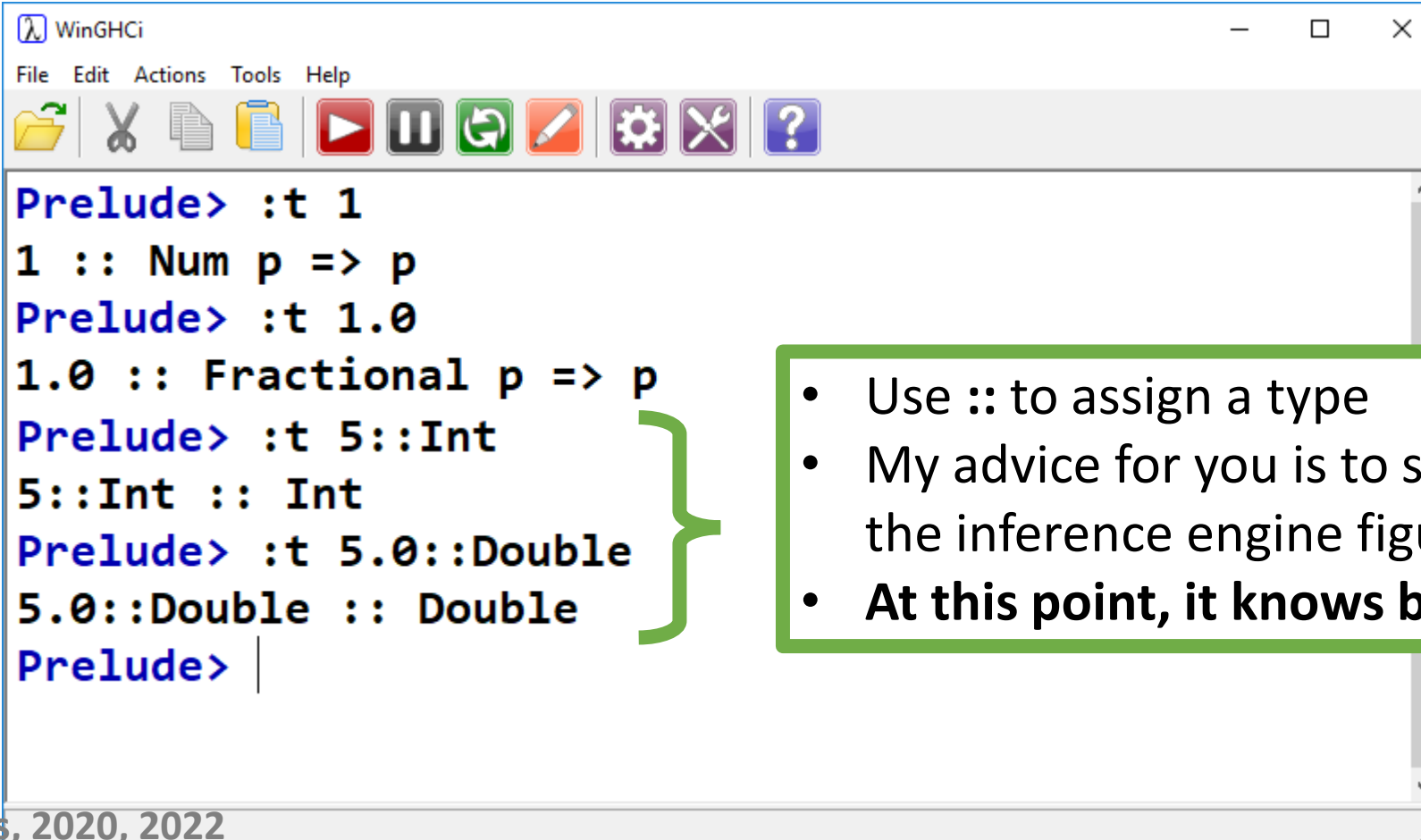
Prelude> :t 1
1 :: Num p => p
Prelude> :t 1.0
1.0 :: Fractional p => p
Prelude> :t 'a'
'a' :: Char
Prelude> :t "Hello"
"Hello" :: [Char]
Prelude> :t 1 > 2
1 > 2 :: Bool
Prelude> |
```

Haskell tries to keep types as generic as possible

- If we explicitly declare a variable as integer, it can't be passed to a function requiring float.
- However, if we generically infer it to be a **Num**, it can be used anywhere any other member of Num is allowed.

Types in Haskell

We can explicitly indicate types:



```
WinGHCi
File Edit Actions Tools Help
[Icons]
Prelude> :t 1
1 :: Num p => p
Prelude> :t 1.0
1.0 :: Fractional p => p
Prelude> :t 5::Int
5::Int :: Int
Prelude> :t 5.0::Double
5.0::Double :: Double
Prelude> |
```

A screenshot of the WinGHCi Haskell interpreter window. The window has a title bar 'WinGHCi' and a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations and execution. The main text area shows the following interactions:

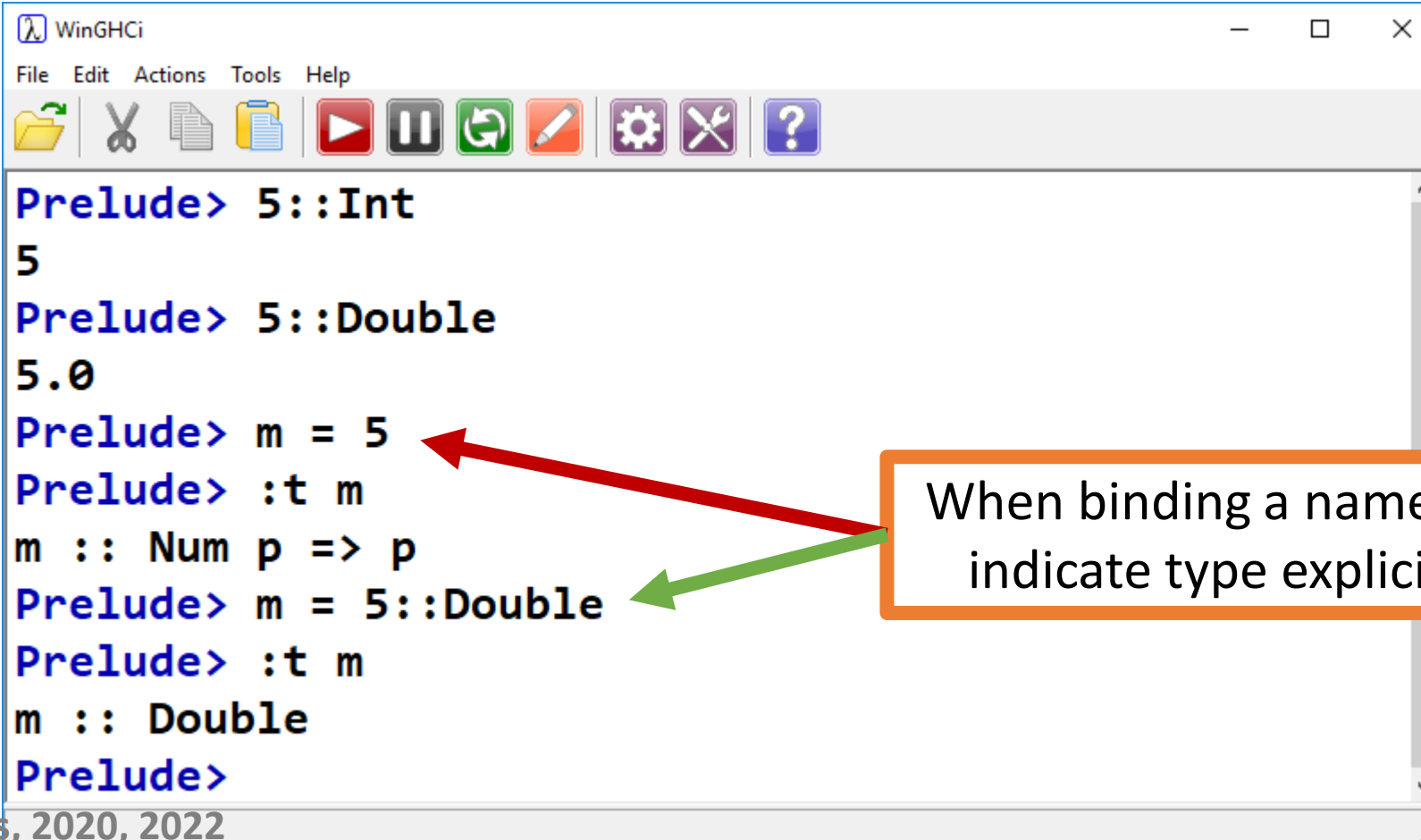
- `Prelude> :t 1` results in `1 :: Num p => p`
- `Prelude> :t 1.0` results in `1.0 :: Fractional p => p`
- `Prelude> :t 5::Int` results in `5::Int :: Int`
- `Prelude> :t 5.0::Double` results in `5.0::Double :: Double`

The last line shows the prompt `Prelude> |` with a cursor. A green bracket on the right side of the last three lines groups them together.

- Use `::` to assign a type
- My advice for you is to start by letting the inference engine figure it out.
- **At this point, it knows better than you.**

Types in Haskell

We can explicitly indicate types:



```
WinGHCi
File Edit Actions Tools Help
[Icons]

Prelude> 5::Int
5
Prelude> 5::Double
5.0
Prelude> m = 5
Prelude> :t m
m :: Num p => p
Prelude> m = 5::Double
Prelude> :t m
m :: Double
Prelude>
```

When binding a name, can indicate type explicitly:

Type Classes

Type polymorphism and type variables:

Recall: Overloading

- In languages like C++, the `==` operator is overloaded to work with many different types.
- Numeric type equality and string equality are performed differently.
- In general, if we want to compare two values of type α , we use an ***α -compare***
- α is a *type variable*, because its value is a type.

Type Classes

Consider the equality (==) operator:

Takes two arguments, each of the same type (call it α), and returns a Boolean

This operator may not be defined for *all* types, just some.

Thus, we can associate == with a specific ***type class*** containing those types for which == is defined.

This type class is called **Eq** in Haskell.

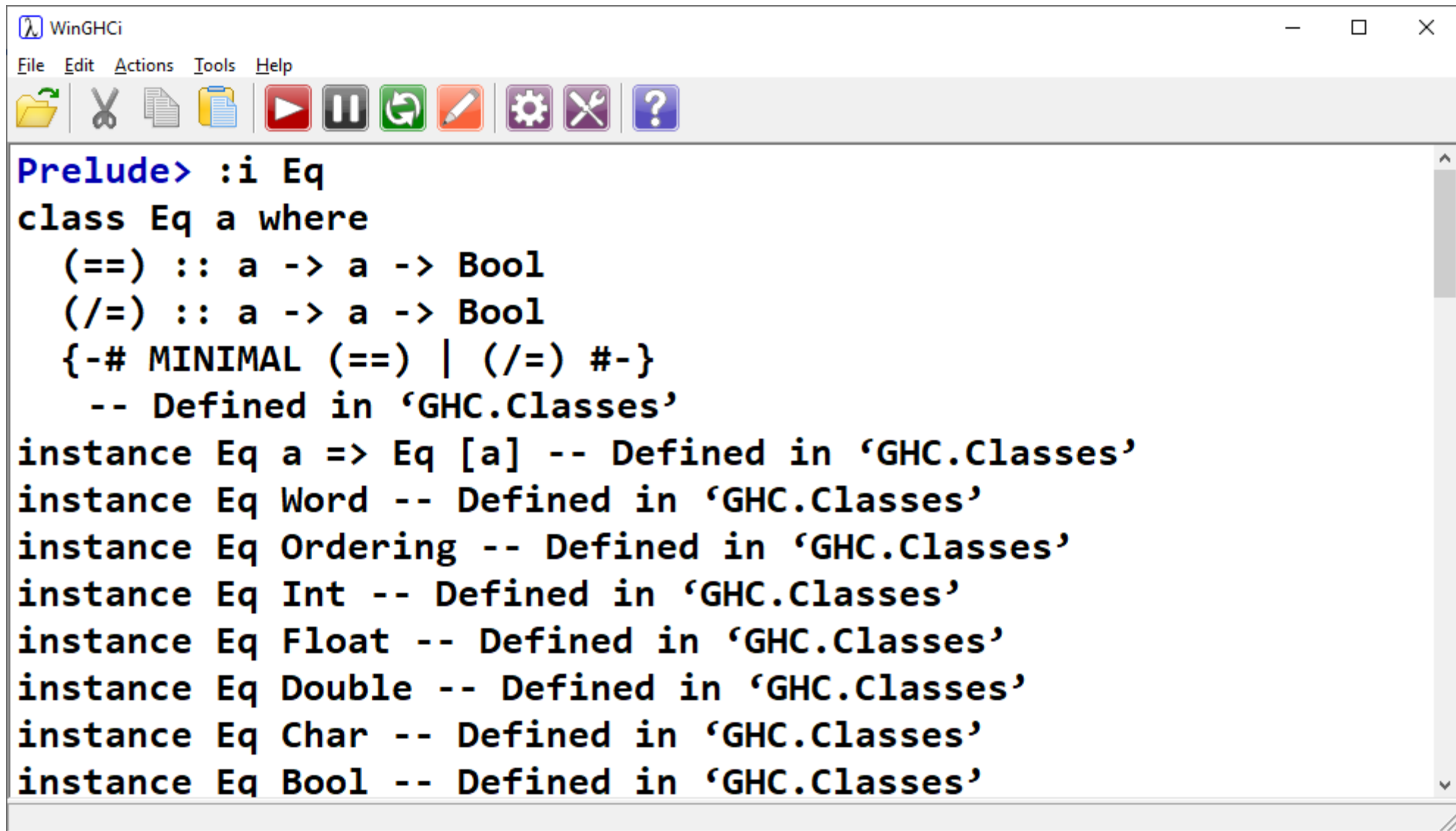
Eq Type Class

`(==)` is defined for types
in typeclass **Eq**

```
(==) :: Eq a => a -> a -> Bool
```

- `(==)` takes two args of type **a**, where **a** is a member of type class **Eq**
- It returns **Bool**

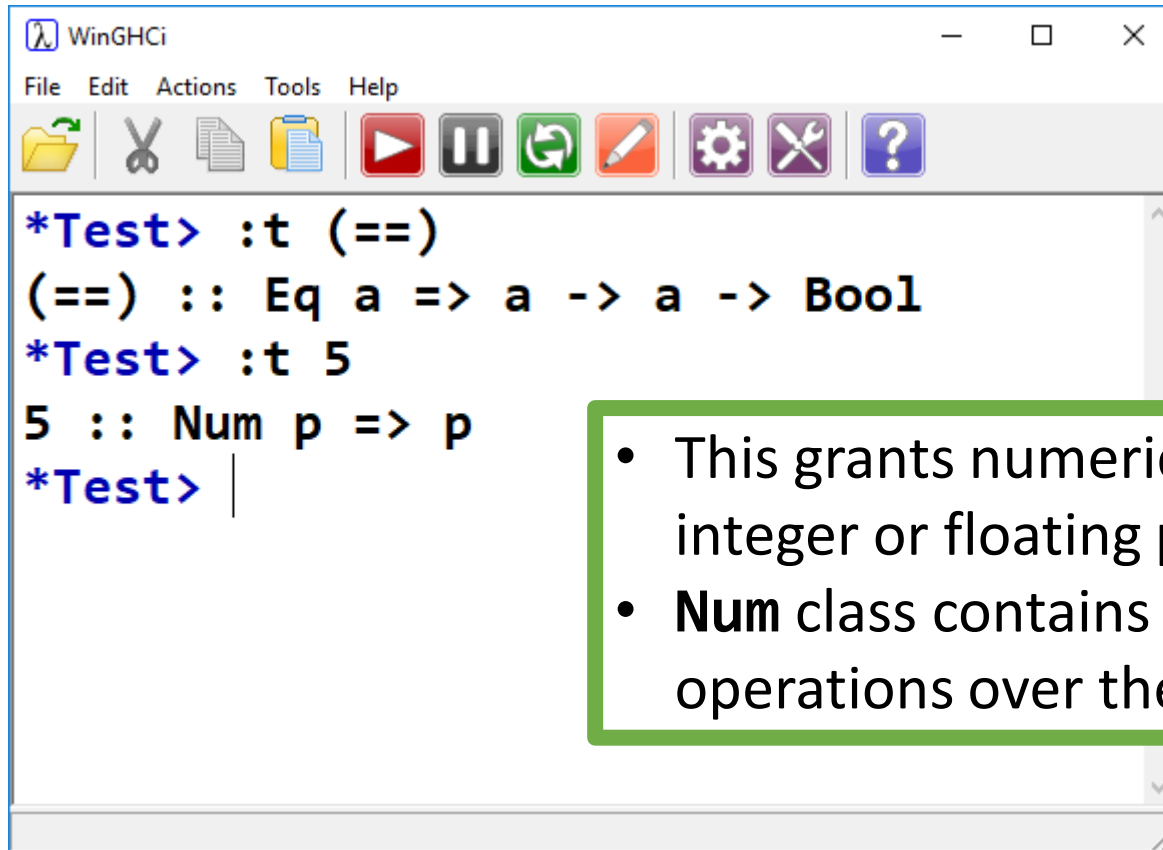
- If a concrete type, **a**, belongs to a certain type class, we say **a** is an *instance* of that type class.
- **Int** is an instance of **Eq**, for example.



The image shows a screenshot of the WinGHCi window. The title bar says 'WinGHCi'. The menu bar includes 'File', 'Edit', 'Actions', 'Tools', and 'Help'. The toolbar contains icons for file operations (copy, paste, save, undo, redo), execution (run, pause, stop), and help (question mark). The main text area displays the following Haskell code:

```
Prelude> :i Eq
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
  {-# MINIMAL (==) | (/=) #-}
  -- Defined in 'GHC.Classes'
instance Eq a => Eq [a] -- Defined in 'GHC.Classes'
instance Eq Word -- Defined in 'GHC.Classes'
instance Eq Ordering -- Defined in 'GHC.Classes'
instance Eq Int -- Defined in 'GHC.Classes'
instance Eq Float -- Defined in 'GHC.Classes'
instance Eq Double -- Defined in 'GHC.Classes'
instance Eq Char -- Defined in 'GHC.Classes'
instance Eq Bool -- Defined in 'GHC.Classes'
```

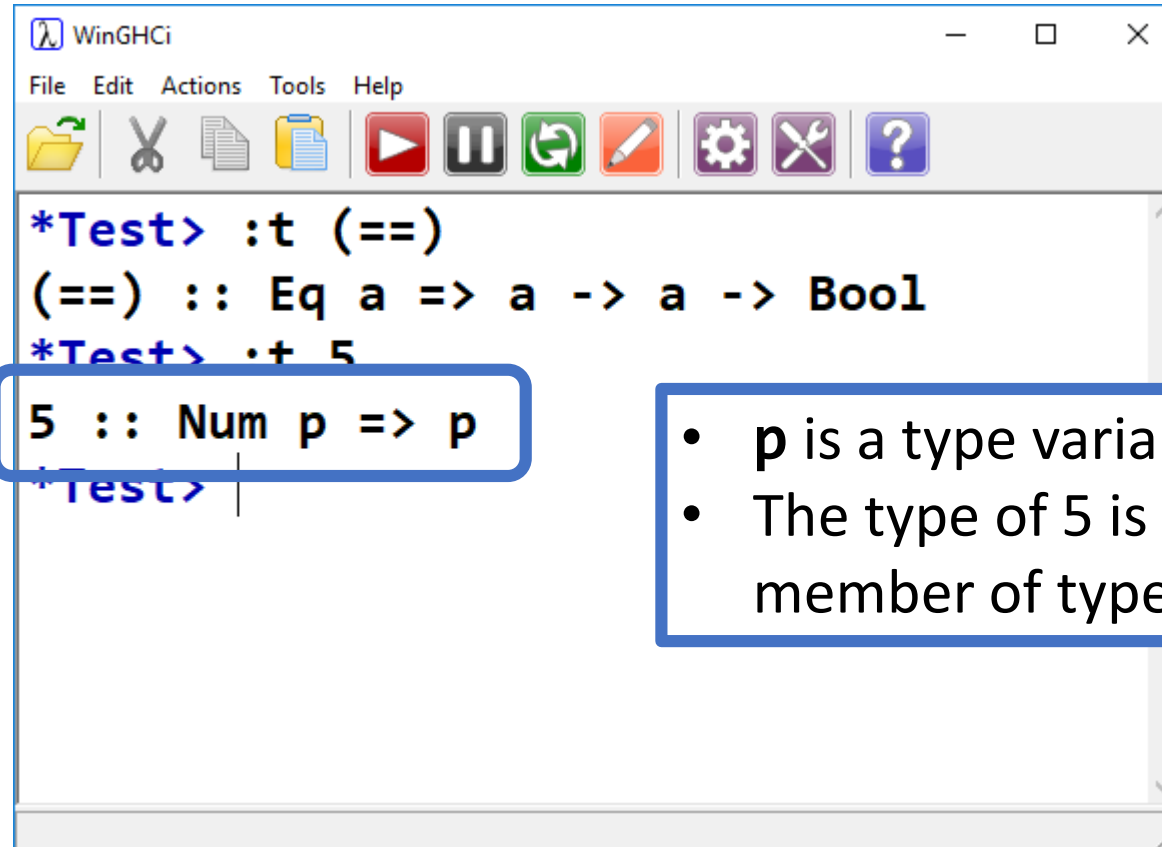
Num Type Class



```
WinGHCi
File Edit Actions Tools Help
[Icons]
*Test> :t (==)
(==) :: Eq a => a -> a -> Bool
*Test> :t 5
5 :: Num p => p
*Test> |
```

- This grants numeric values freedom to be an integer or floating point as the compiler sees fit.
- **Num** class contains all numbers, and certain operations over them such as addition.

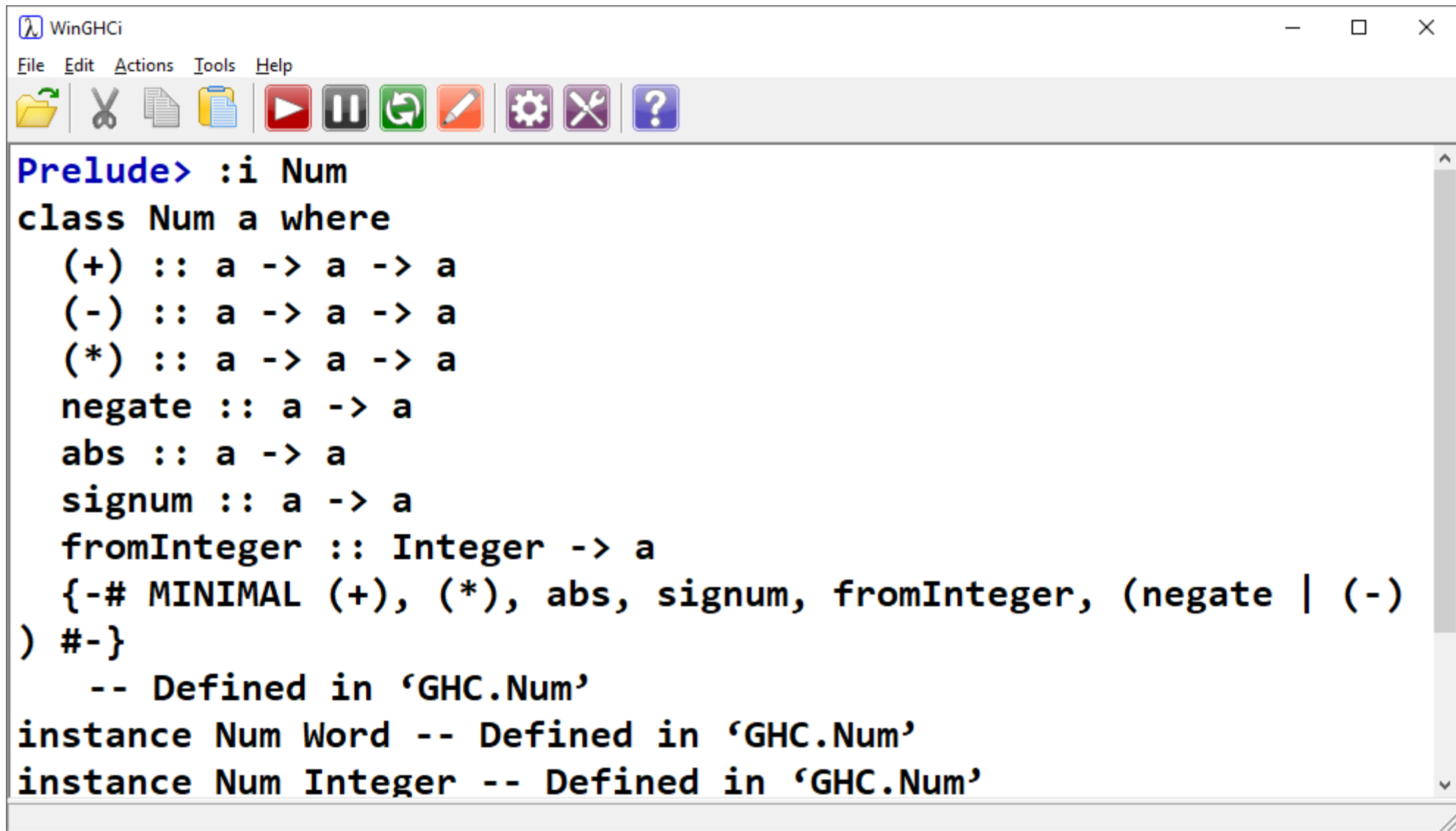
Num Type Class

A screenshot of the WinGHCi window. The window has a title bar 'WinGHCi' and a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (folder, scissors, copy, paste), execution (play, pause, refresh), editing (pencil), settings (gear), and help (question mark). The main text area contains the following code:

```
*Test> :t (==)
(==) :: Eq a => a -> a -> Bool
*Test> :t 5
5 :: Num p => p
*Test>
```

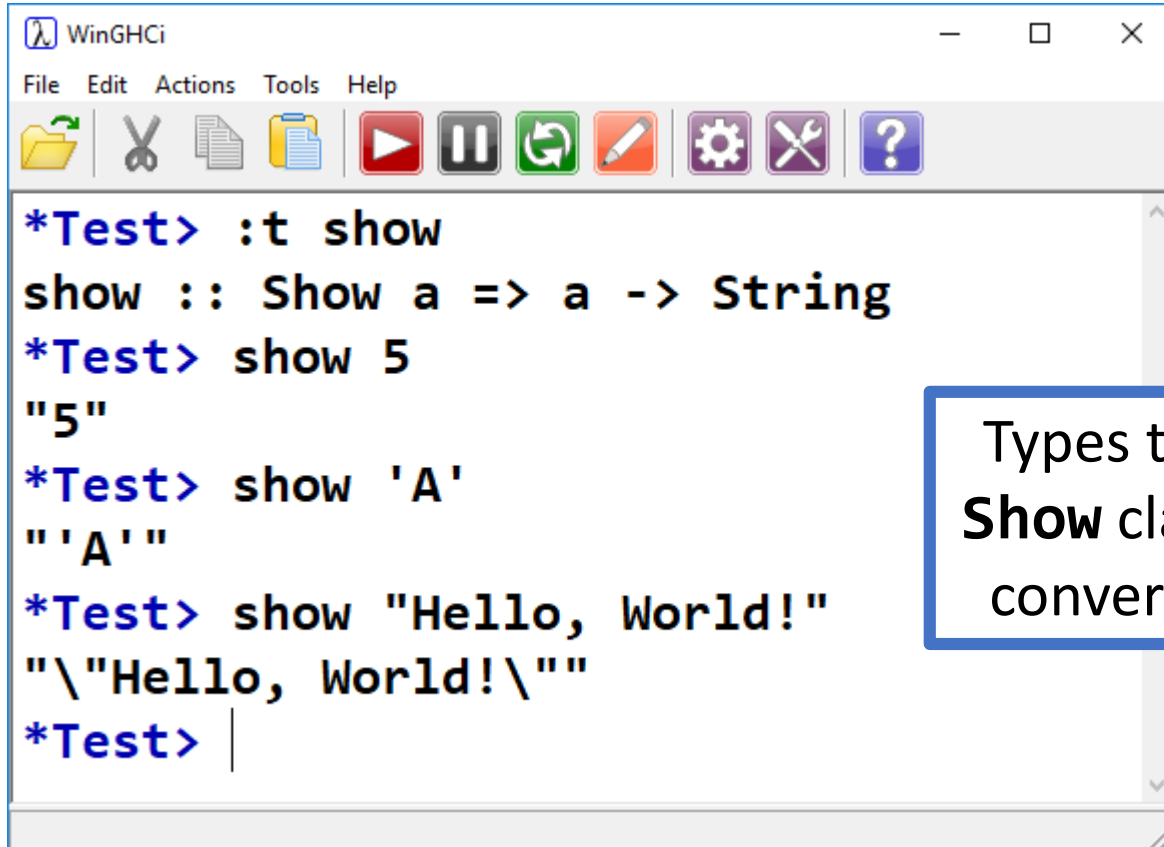
The line `5 :: Num p => p` is highlighted with a blue box.

- **p** is a type variable
- The type of 5 is **p**, and **p** is a member of type class **Num**

A screenshot of a WinGHCi window. The window has a title bar with the text 'WinGHCi' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Under the 'Actions' menu, there is a toolbar with icons for file operations (open, save, copy, paste), execution (run, pause, refresh), and help (question mark). The main text area contains Haskell code defining the 'Num' class and its instances. The code is as follows:

```
Prelude> :i Num
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)
) #-}
  -- Defined in 'GHC.Num'
instance Num Word -- Defined in 'GHC.Num'
instance Num Integer -- Defined in 'GHC.Num'
```

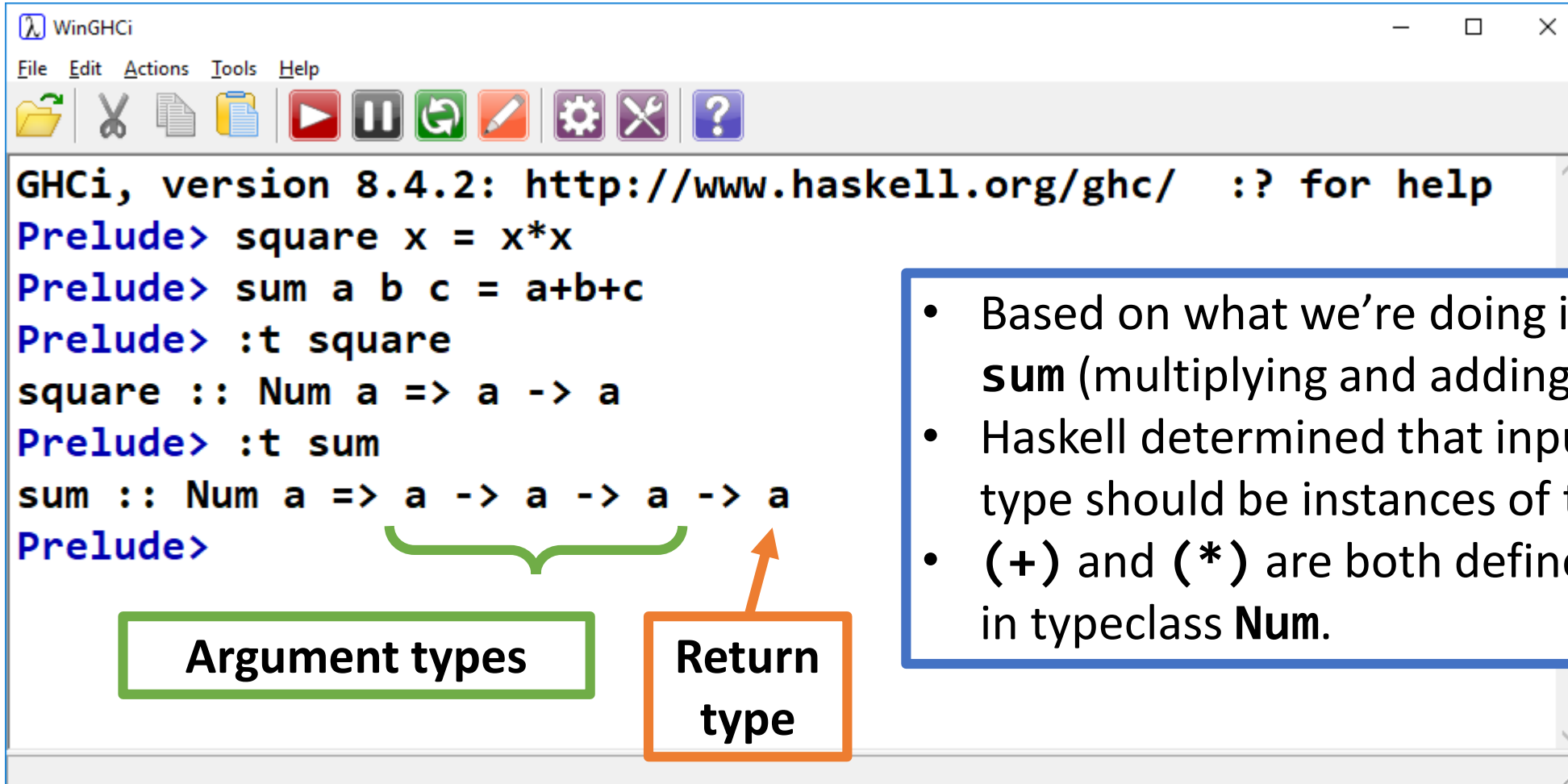
Show Type Class

A screenshot of the WinGHCi terminal window. The window has a title bar with the WinGHCi logo and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Underneath the menu bar is a toolbar with icons for file operations (folder, copy, paste, save), execution (play, pause, refresh), editing (undo, redo, delete), and settings (gear, wrench, question mark). The main area of the window is a text editor showing the following Haskell code:

```
*Test> :t show
show :: Show a => a -> String
*Test> show 5
"5"
*Test> show 'A'
"'A'"
*Test> show "Hello, World!"
 "\"Hello, World!\""
*Test> |
```

Types that are members of the **Show** class have functions which convert their value to a String.

Functions & Typeclasses



WinGHCi

File Edit Actions Tools Help

GHCi, version 8.4.2: <http://www.haskell.org/ghc/> :? for help

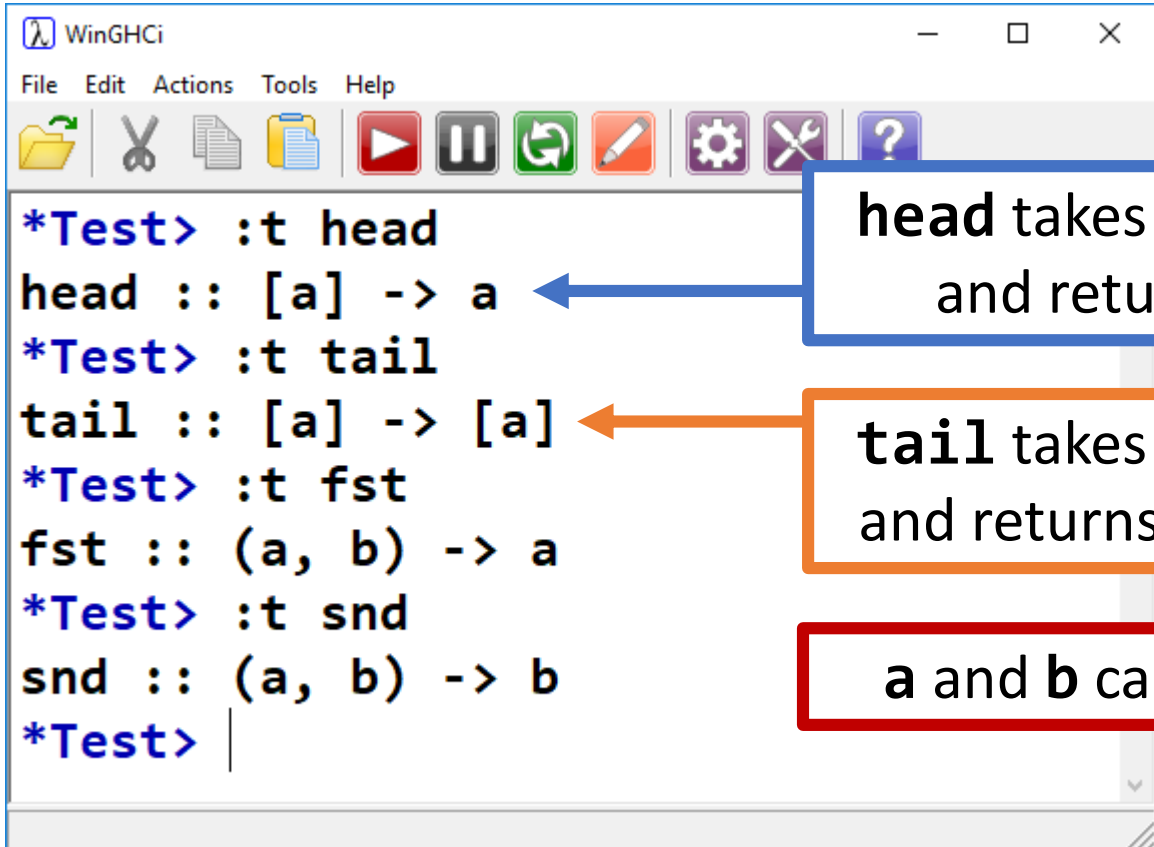
```
Prelude> square x = x*x
Prelude> sum a b c = a+b+c
Prelude> :t square
square :: Num a => a -> a
Prelude> :t sum
sum :: Num a => a -> a -> a -> a
Prelude>
```

Argument types

Return type

- Based on what we're doing in **square** and **sum** (multiplying and adding)...
- Haskell determined that input and output type should be instances of typeclass **Num**.
- **(+)** and **(*)** are both defined for all types in typeclass **Num**.

Function Type Signatures



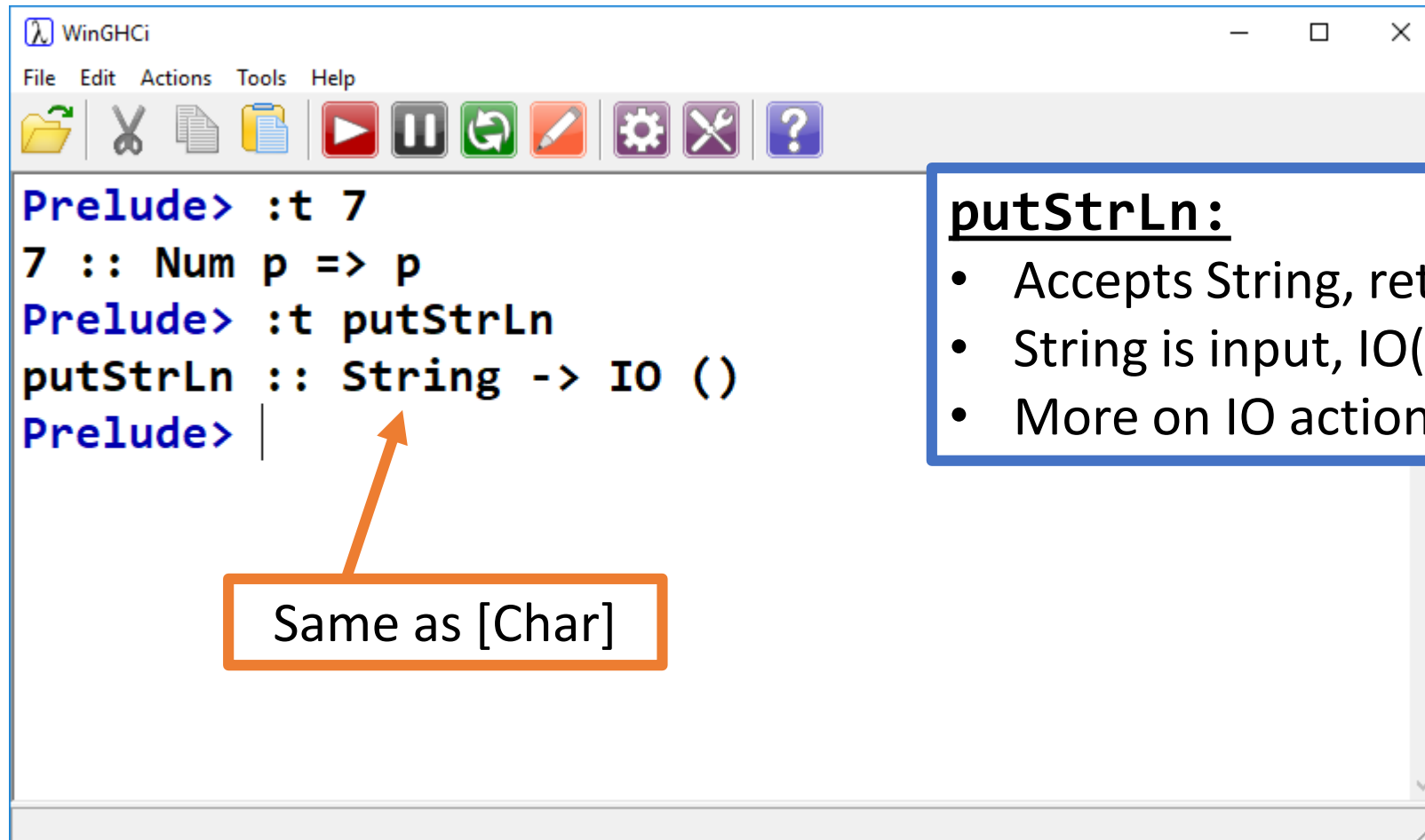
The image shows a WinGHCi terminal window with the following content:

```
*Test> :t head
head :: [a] -> a
*Test> :t tail
tail :: [a] -> [a]
*Test> :t fst
fst :: (a, b) -> a
*Test> :t snd
snd :: (a, b) -> b
*Test> |
```

Callouts explaining the functions:

- head** takes a list containing type **a**, and returns a value of type **a**
- tail** takes a list containing type **a**, and returns a list containing type **a**
- a** and **b** can be *literally any type*!

Function Type Signatures



The image shows a screenshot of the WinGHCi window. The window has a title bar 'WinGHCi' and a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (open, save, copy, paste), execution (run, stop, refresh), and editing (undo, redo, settings, help). The main text area contains the following Haskell code:

```
Prelude> :t 7
7 :: Num p => p
Prelude> :t putStrLn
putStrLn :: String -> IO ()
Prelude> |
```

An orange arrow points from a box labeled 'Same as [Char]' to the 'String' type in the signature of 'putStrLn'.

putStrLn:

- Accepts String, returns ***IO action***.
- String is input, IO() is output.
- More on IO actions later.

Same as [Char]

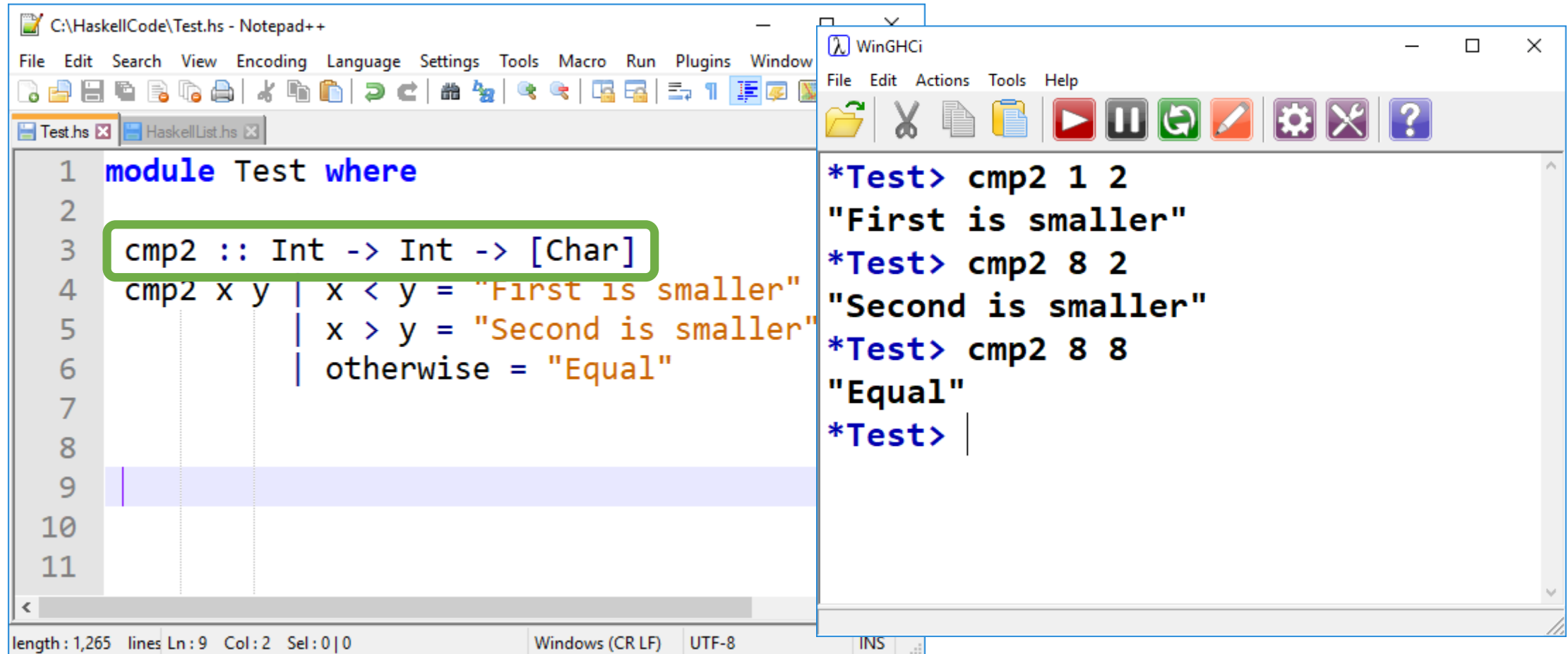
Specify Function Type

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 chkAxis :: (Float, Float) -> (Float, Float)
4 chkAxis (0, _) = (0, 1)
5 chkAxis (_, 0) = (1, 0)
6 chkAxis (a, b) = (a, b)
7
```

- **chkAxis** takes a pair-tuple of Floats as input, and returns the same as output.
- Instead of constants being of type Num or Fractional, they are treated as Floats

```
WinGHCi
File Edit Actions Tools Help
*Test> chkAxis (1, 0)
(1.0,0.0)
*Test> chkAxis (0, 4.5)
(0.0,1.0)
*Test> chkAxis (3, 4)
(3.0,4.0)
*Test> chkAxis (4.333, 0)
(1.0,0.0)
*Test> :t chkAxis
chkAxis :: (Float, Float) ->
(Float, Float)
*Test>
```

Specify Function Type

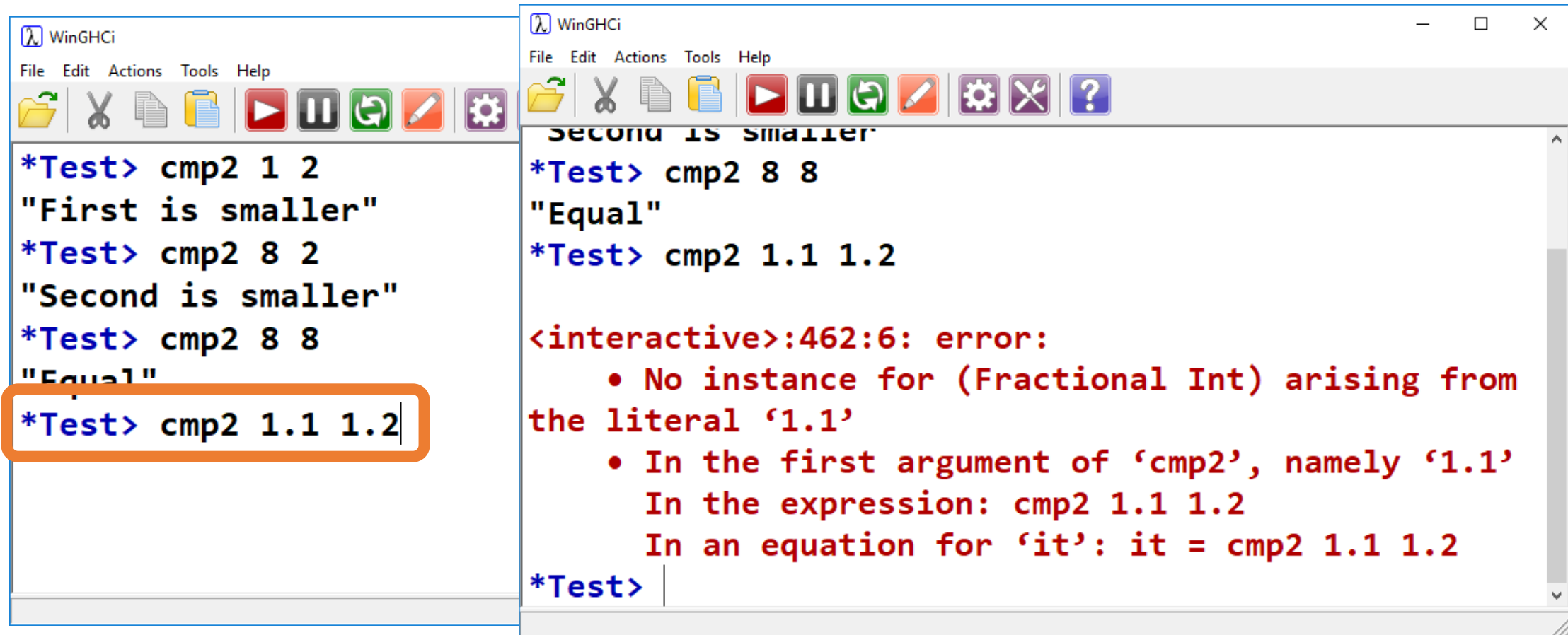


The image shows two windows side-by-side. The left window is Notepad++ editing a file named 'Test.hs'. The code defines a module 'Test' with a function 'cmp2' that takes two integers and returns a list of characters. The function's type signature 'cmp2 :: Int -> Int -> [Char]' is highlighted with a green box. The function body uses pattern matching to return 'First is smaller', 'Second is smaller', or 'Equal' based on the comparison of the two integers. The right window is WinGHCi, which shows the execution of the 'cmp2' function with three test cases: 'cmp2 1 2' returns 'First is smaller', 'cmp2 8 2' returns 'Second is smaller', and 'cmp2 8 8' returns 'Equal'.

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
Test.hs HaskellList.hs
1 module Test where
2
3 cmp2 :: Int -> Int -> [Char]
4 cmp2 x y | x < y = "First is smaller"
5           | x > y = "Second is smaller"
6           | otherwise = "Equal"
7
8
9
10
11
length: 1,265 lines Ln: 9 Col: 2 Sel: 0|0 Windows (CR LF) UTF-8

WinGHCi
File Edit Actions Tools Help
*Test> cmp2 1 2
"First is smaller"
*Test> cmp2 8 2
"Second is smaller"
*Test> cmp2 8 8
"Equal"
*Test> |
```


Thoughts?



The image displays two side-by-side screenshots of the WinGHCi Haskell interpreter interface. The left window shows a series of successful comparisons using the `cmp2` function. The right window shows an error message when comparing a float with an integer.

Left Window (WinGHCi):

```
*Test> cmp2 1 2
"First is smaller"
*Test> cmp2 8 2
"Second is smaller"
*Test> cmp2 8 8
"Equal"
*Test> cmp2 1.1 1.2
```

Right Window (WinGHCi):

```
second is smaller
*Test> cmp2 8 8
"Equal"
*Test> cmp2 1.1 1.2

<interactive>:462:6: error:
    • No instance for (Fractional Int) arising from
      the literal '1.1'
    • In the first argument of 'cmp2', namely '1.1'
      In the expression: cmp2 1.1 1.2
      In an equation for 'it': it = cmp2 1.1 1.2
*Test>
```

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 --cmp2 :: Int -> Int -> [Char]
4 cmp2 x y | x < y = "First is smaller"
5          | x > y = "Second is smaller"
6          | otherwise = "Equal"
7
8
9
10
11
length: 1,267 lines Ln: 9 Col: 2 Sel: 0|0
```

```
WinGHCi
File Edit Actions Tools Help
*Test> cmp2 1.1 1.2
"First is smaller"
*Test> :t cmp2
cmp2 :: Ord a => a -> a -> [Char]
*Test>
```

Ord is a type class:

- When we didn't explicitly define our types, Haskell inferred the type for us.
- Ord is a type class under which the operations used on our inputs are defined.
- I.e., comparison operators.

Type VS Type Class

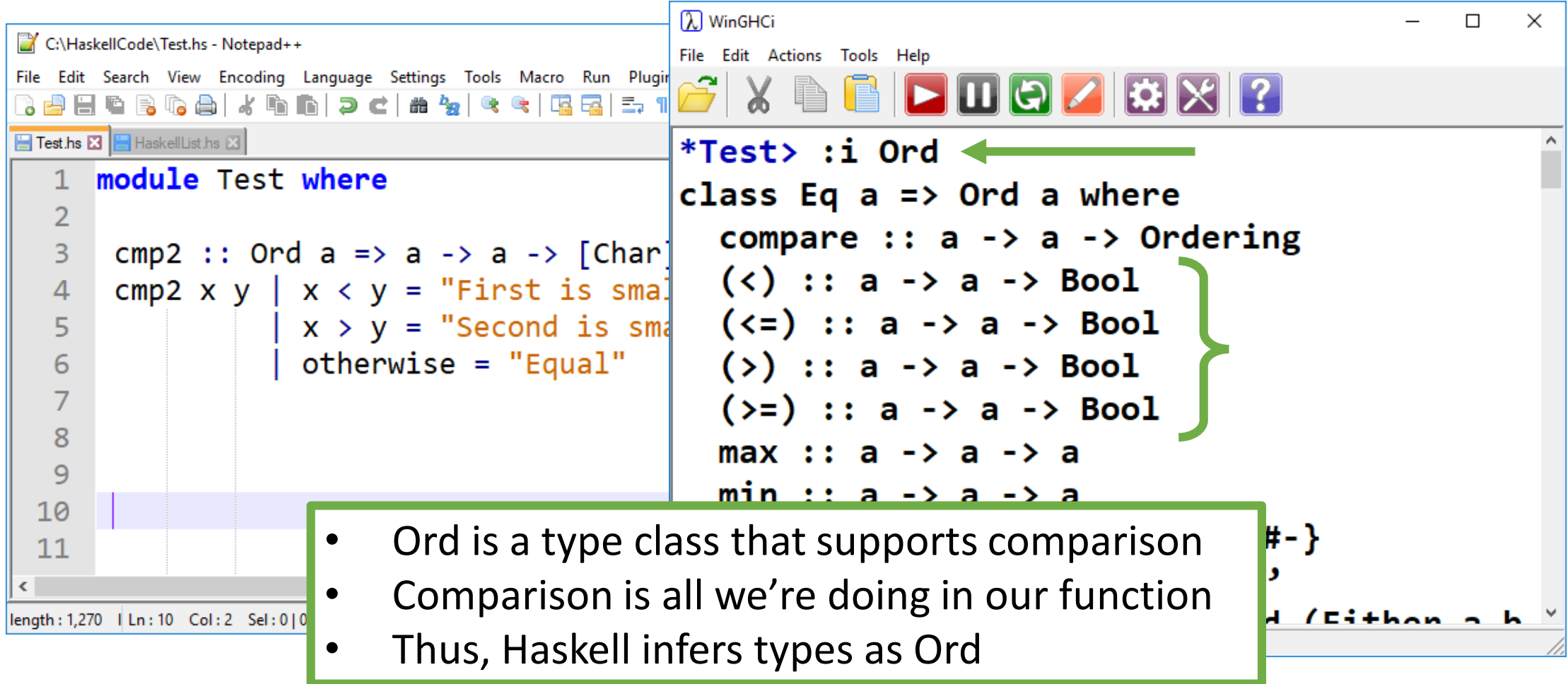
```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
Test.hs HaskellList.hs
1 module Test where
2
3 cmp2 :: Int -> Int -> [Char]
4 cmp2 x y | x < y = "First is smaller"
5          | x > y = "Second is smaller"
6          | otherwise = "Equal"
7
8
9
10
```

- Int & Char are types, **not** type classes
- We can use the above notation

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
Test.hs HaskellList.hs
1 module Test where
2
3 cmp2 :: Ord a => a -> a -> [Char]
4 cmp2 x y | x < y = "First is smaller"
5          | x > y = "Second is smaller"
6          | otherwise = "Equal"
7
8
9
10
```

- Ord is a type class, thus we specify that **a** is an instance of Ord
- **cmp2** accepts two instances of Ord as arguments.
- Ord contains many different types, **a** can be any of them

Ord Type Class



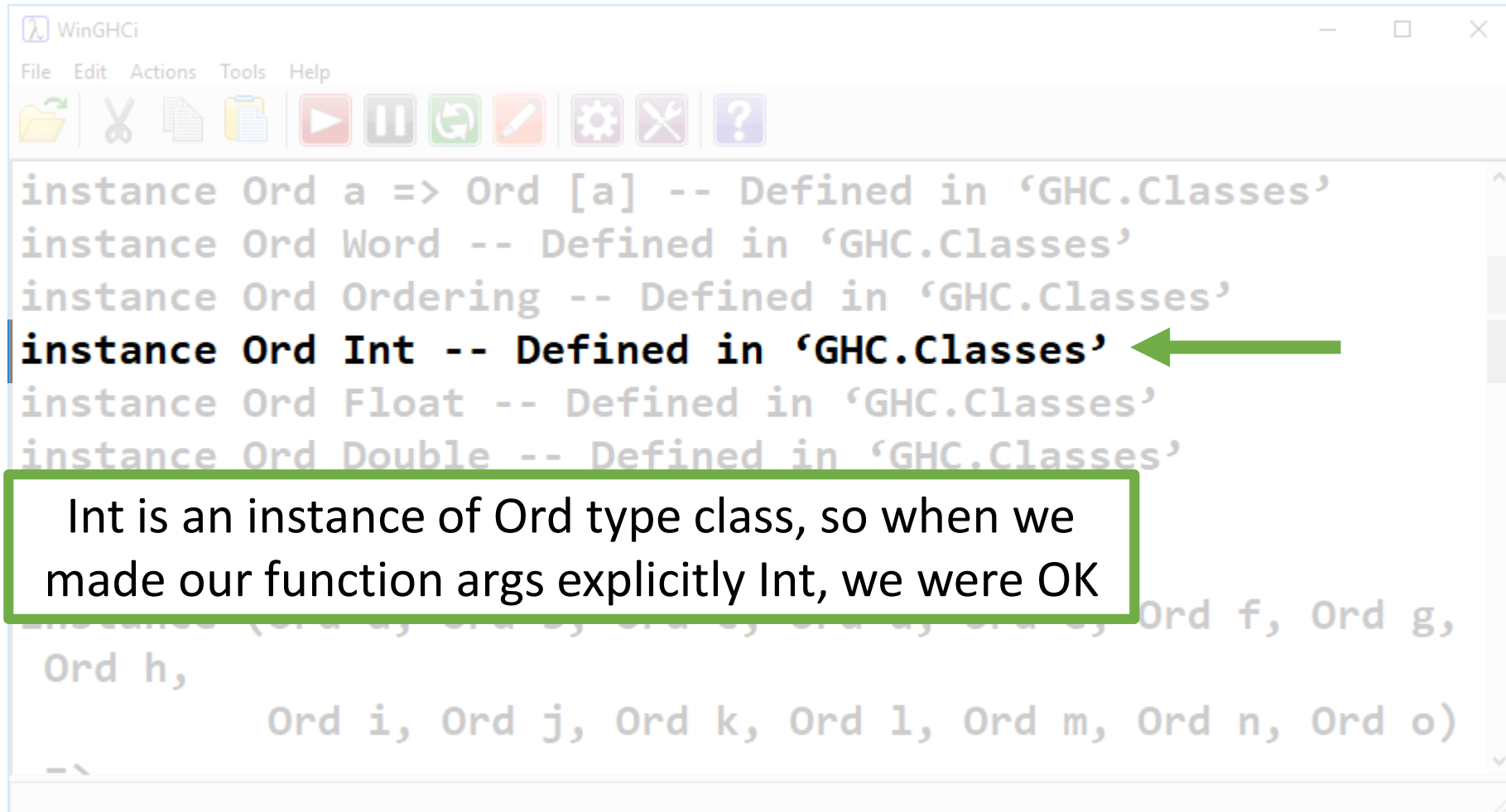
The image shows two windows. The left window, Notepad++, displays a Haskell module named 'Test' with a function 'cmp2' that takes two arguments of type 'Ord a' and returns a list of characters. The function uses comparisons like 'x < y' and 'x > y' to determine the order. The right window, WinGHCi, shows the interactive prompt '*Test> :i Ord' with a green arrow pointing to it. Below the prompt, the definition of the 'Ord' type class is shown, including the 'Eq' class and the 'compare' function. A green bracket groups the comparison functions: '(<)', '(<=)', '(>)', and '(>=)'. A green box at the bottom contains a list of bullet points explaining the 'Ord' type class.

```
1 module Test where
2
3 cmp2 :: Ord a => a -> a -> [Char]
4 cmp2 x y | x < y = "First is small"
5          | x > y = "Second is small"
6          | otherwise = "Equal"
7
8
9
10
11
```

```
*Test> :i Ord
class Eq a => Ord a where
  compare :: a -> a -> Ordering
  (<) :: a -> a -> Bool
  (<=) :: a -> a -> Bool
  (>) :: a -> a -> Bool
  (>=) :: a -> a -> Bool
  max :: a -> a -> a
  min :: a -> a -> a
```

- Ord is a type class that supports comparison
- Comparison is all we're doing in our function
- Thus, Haskell infers types as Ord

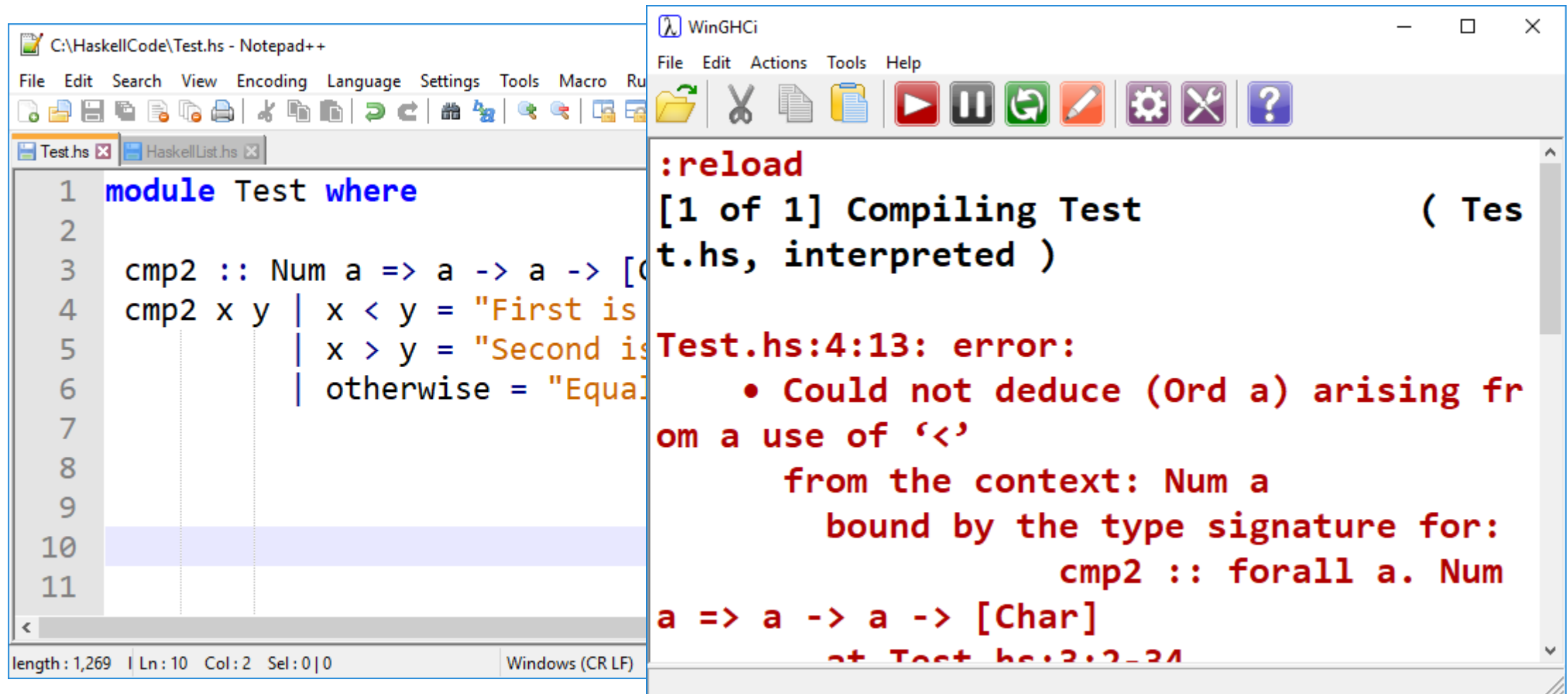
Ord Type Class



```
WinGHCi
File Edit Actions Tools Help
instance Ord a => Ord [a] -- Defined in 'GHC.Classes'
instance Ord Word -- Defined in 'GHC.Classes'
instance Ord Ordering -- Defined in 'GHC.Classes'
instance Ord Int -- Defined in 'GHC.Classes'
instance Ord Float -- Defined in 'GHC.Classes'
instance Ord Double -- Defined in 'GHC.Classes'
instance (Ord a, Ord b, Ord c, Ord d, Ord e, Ord f, Ord g,
Ord h,
Ord i, Ord j, Ord k, Ord l, Ord m, Ord n, Ord o)
--
```

Int is an instance of Ord type class, so when we made our function args explicitly Int, we were OK

How About This?

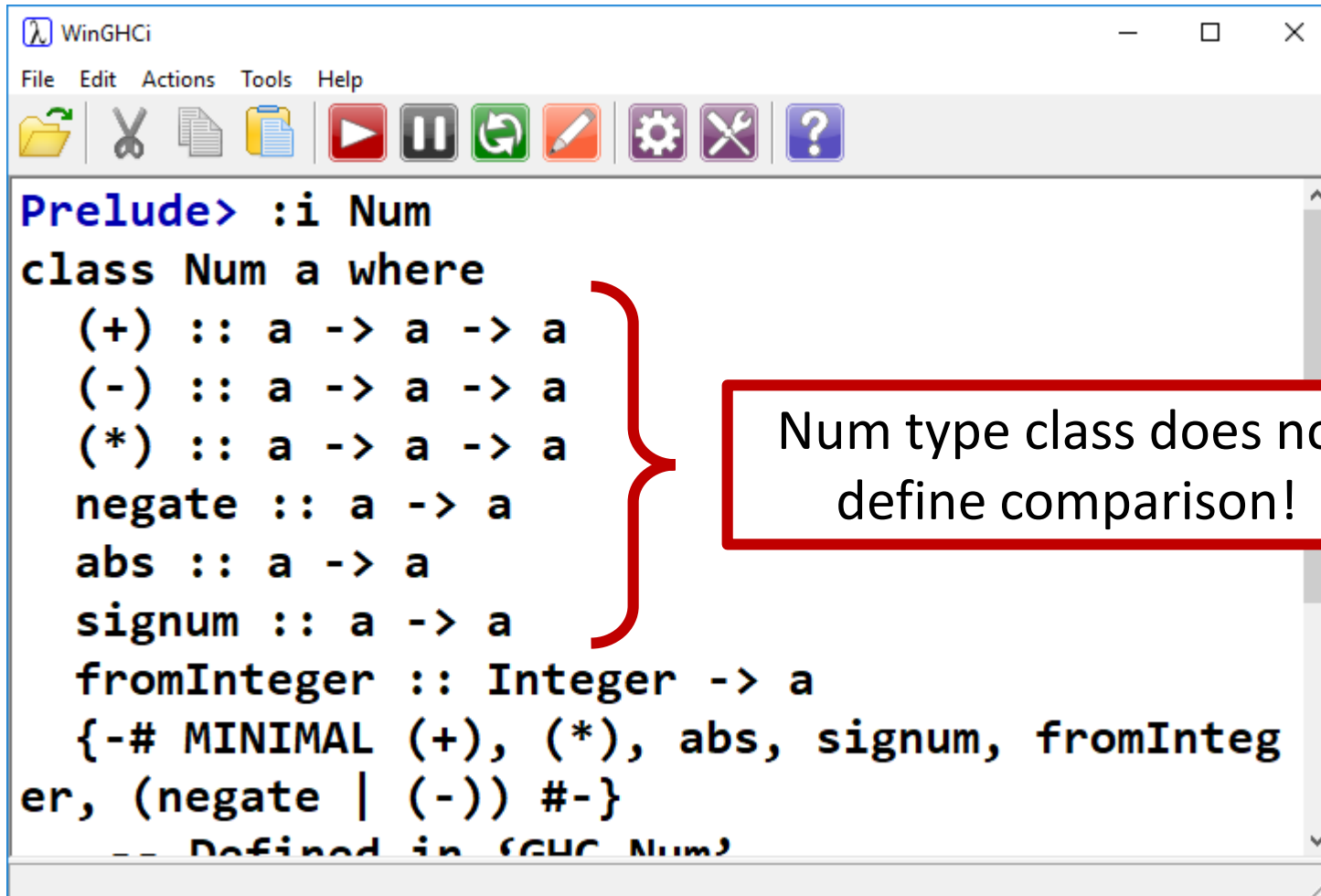


The image shows a Haskell development environment. On the left is a Notepad++ window titled 'C:\HaskellCode\Test.hs - Notepad++'. It contains the following Haskell code:

```
1 module Test where
2
3 cmp2 :: Num a => a -> a -> [Char]
4 cmp2 x y | x < y = "First is smaller"
5          | x > y = "Second is smaller"
6          | otherwise = "Equal"
7
8
9
10
11
```

On the right is a WinGHCi window. It shows the command `:reload` and the output `[1 of 1] Compiling Test.hs, interpreted`. Below this, a red error message is displayed:

```
Test.hs:4:13: error:
    • Could not deduce (Ord a) arising from
      a use of '<'
        from the context: Num a
           bound by the type signature for:
               cmp2 :: forall a. Num
                  a => a -> a -> [Char]
```



```
WinGHCi
File Edit Actions Tools Help
[Icons]

Prelude> :i Num
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)) #-}
  -- Defined in (GHC.Num)
```

Num type class does not
define comparison!

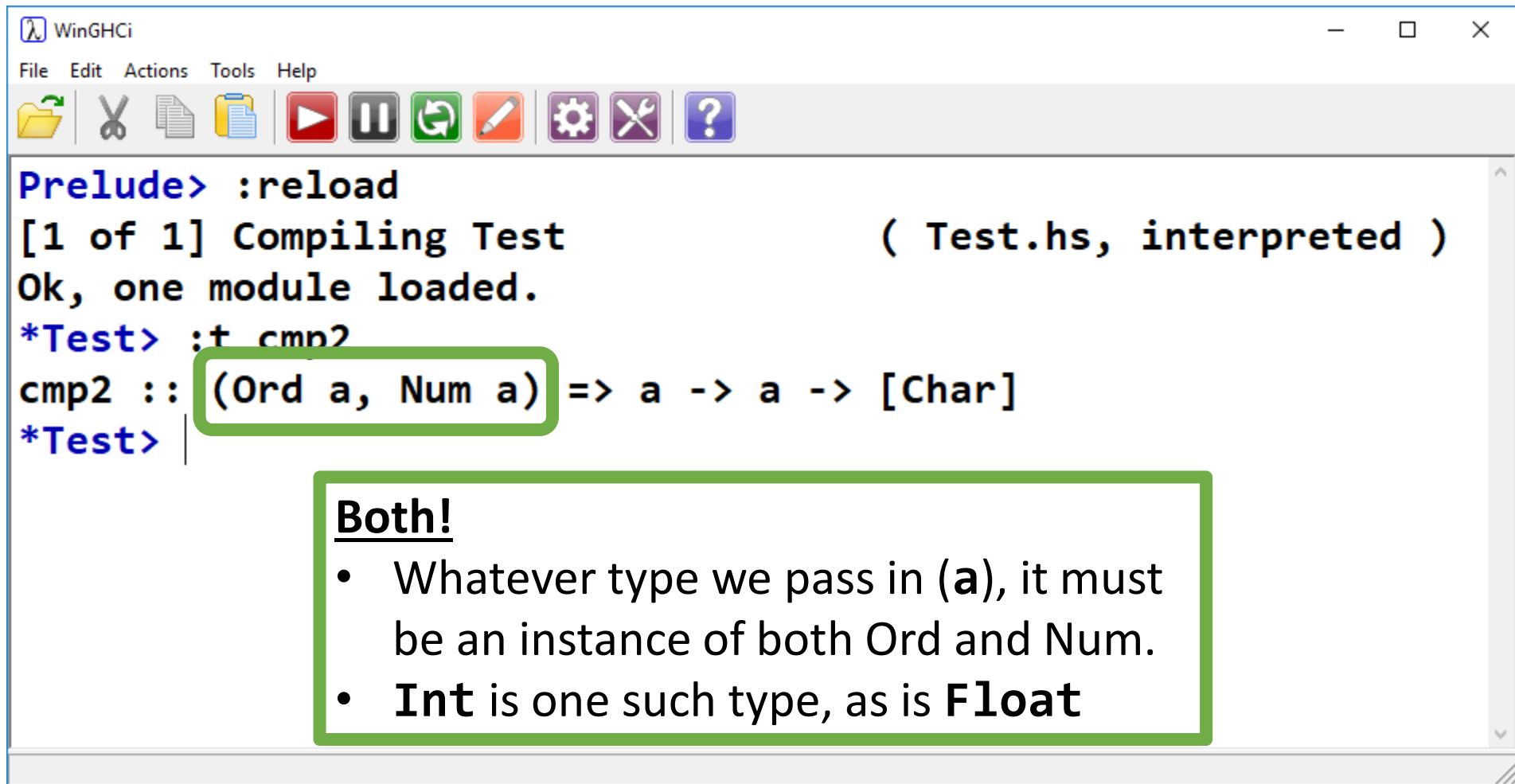
Hmmm...

Num doesn't have comparison, **Ord** doesn't have addition

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 cmp2 x y | x+1 < y+1 = "First is smaller"
4          | x+1 > y+1 = "Second is smaller"
5          | otherwise = "Equal"
6
7
8
9
10
11
length: 1,241 lines: 77 Ln: 10 Col: 2 Sel: 0|0 Windows (CR LF) UTF-8
```

```
WinGHCi
File Edit Actions Tools Help
Prelude> :reload
[1 of 1] Compiling Test
( Test.hs, interpreted )
Ok, one module loaded.
*Test> |
```

It compiled and loaded, what type did Haskell infer for x and y?



The image shows a screenshot of the WinGHCi window. The window has a title bar with the text 'WinGHCi' and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Under the menu bar is a toolbar with icons for file operations (folder, scissors, document), execution (play, pause, refresh), editing (pencil), settings (gear), and help (question mark). The main text area contains the following Haskell code:

```
Prelude> :reload
[1 of 1] Compiling Test                ( Test.hs, interpreted )
Ok, one module loaded.
*Test> :t cmp2
cmp2 :: (Ord a, Num a) => a -> a -> [Char]
*Test> |
```

A green rectangular box highlights the type signature `(Ord a, Num a) => a -> a -> [Char]`. Below this, another green rectangular box contains the following text:

Both!

- Whatever type we pass in (**a**), it must be an instance of both **Ord** and **Num**.
- **Int** is one such type, as is **Float**

Ord:

```
WinGHCi
File Edit Actions Tools Help
instance Ord Ordering -- Defined in 'GHC.Classes'
instance Ord Int -- Defined in 'GHC.Classes'
instance Ord Float -- Defined in 'GHC.Classes'
instance Ord Double -- Defined in 'GHC.Classes'
instance Ord Char -- Defined in 'GHC.Classes'
instance Ord Bool -- Defined in 'GHC.Classes'
instance (Ord a, Ord b, Ord c, Ord d, Ord e, Ord f, Ord
```

Num:

```
WinGHCi
File Edit Actions Tools Help
-- Defined in 'GHC.Num'
instance Num Word -- Defined in 'GHC.Num'
instance Num Integer -- Defined in 'GHC.Num'
instance Num Int -- Defined in 'GHC.Num'
instance Num Float -- Defined in 'GHC.Float'
instance Num Double -- Defined in 'GHC.Float'
*Test>
```

Custom Data Types

Custom Data Types

- Lists and tuples are already quite powerful for organizing data
- What if we want to add custom behaviors over our data?
- For example, we can declare a pair tuple (1, 2).
- What if we want to treat these as coordinates and compute the sum? The dot product? Etc.?
- Addition is not defined for tuples, let alone more complicated operations.

Custom Coordinate Types

```
data Pt3 = Pt3 Float Float Float
```

Keyword
indicating a
custom type
definition

Custom
type name

Constructor for our custom type.
To construct a Pt3, we need 3
values of type Float

C:\HaskellCode\Test.hs - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X

Test.hs HaskellList.hs

```
1 module Test where
2
3 data Pt3 = Pt3 Float Float Float
4
5
6
7
```

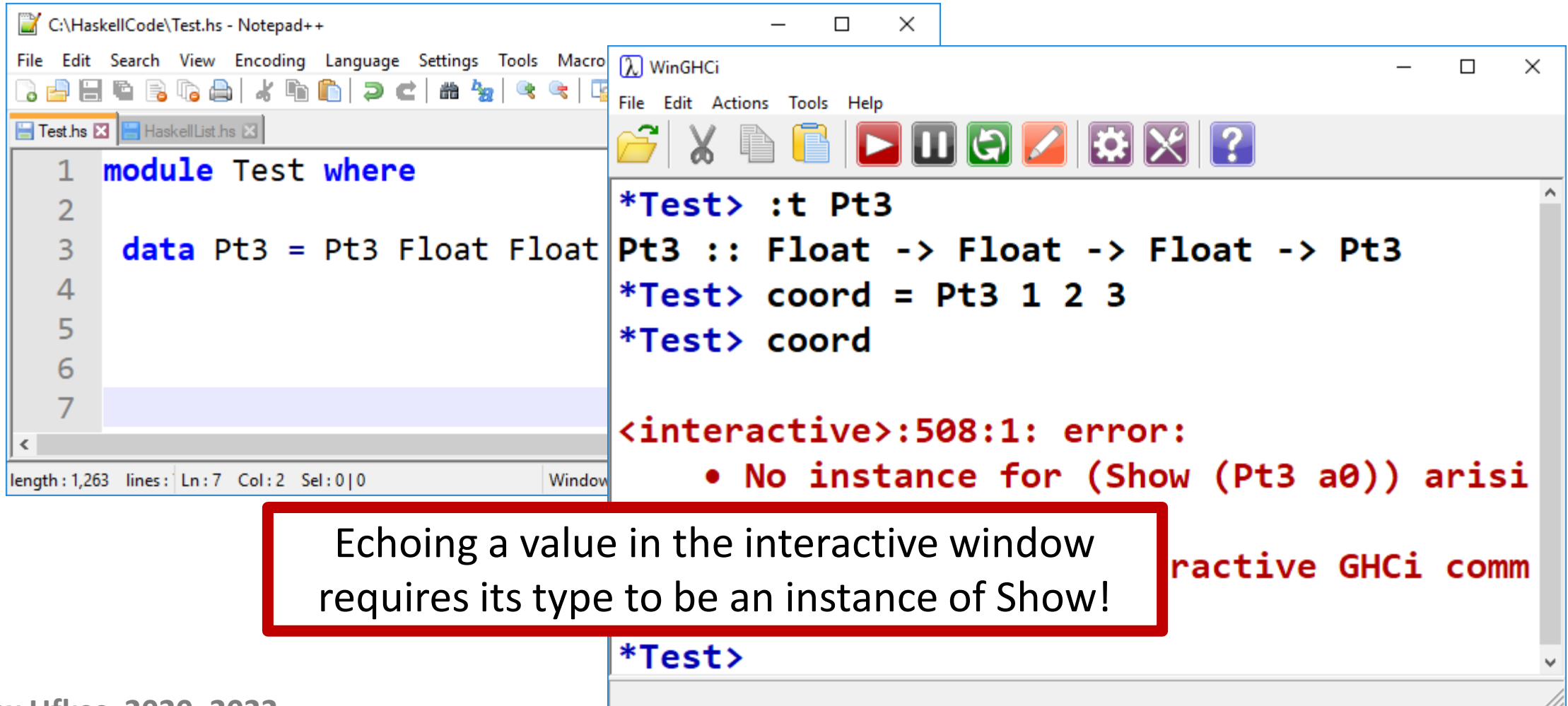
length: 1,263 lines: Ln: 7 Col: 2 Sel: 0|0

WinGHCi

File Edit Actions Tools Help

```
*Test> :t Pt3
Pt3 :: Float -> Float -> Float -> Pt3
*Test> |
```

Custom Type Usage



The image shows two windows. The left window is Notepad++ editing `C:\HaskellCode\Test.hs`. The code is:

```
1 module Test where
2
3 data Pt3 = Pt3 Float Float
4
5
6
7
```

The right window is WinGHCi. The prompt is `*Test>`. The user has entered:

```
:t Pt3
Pt3 :: Float -> Float -> Float -> Pt3
coord = Pt3 1 2 3
coord
```

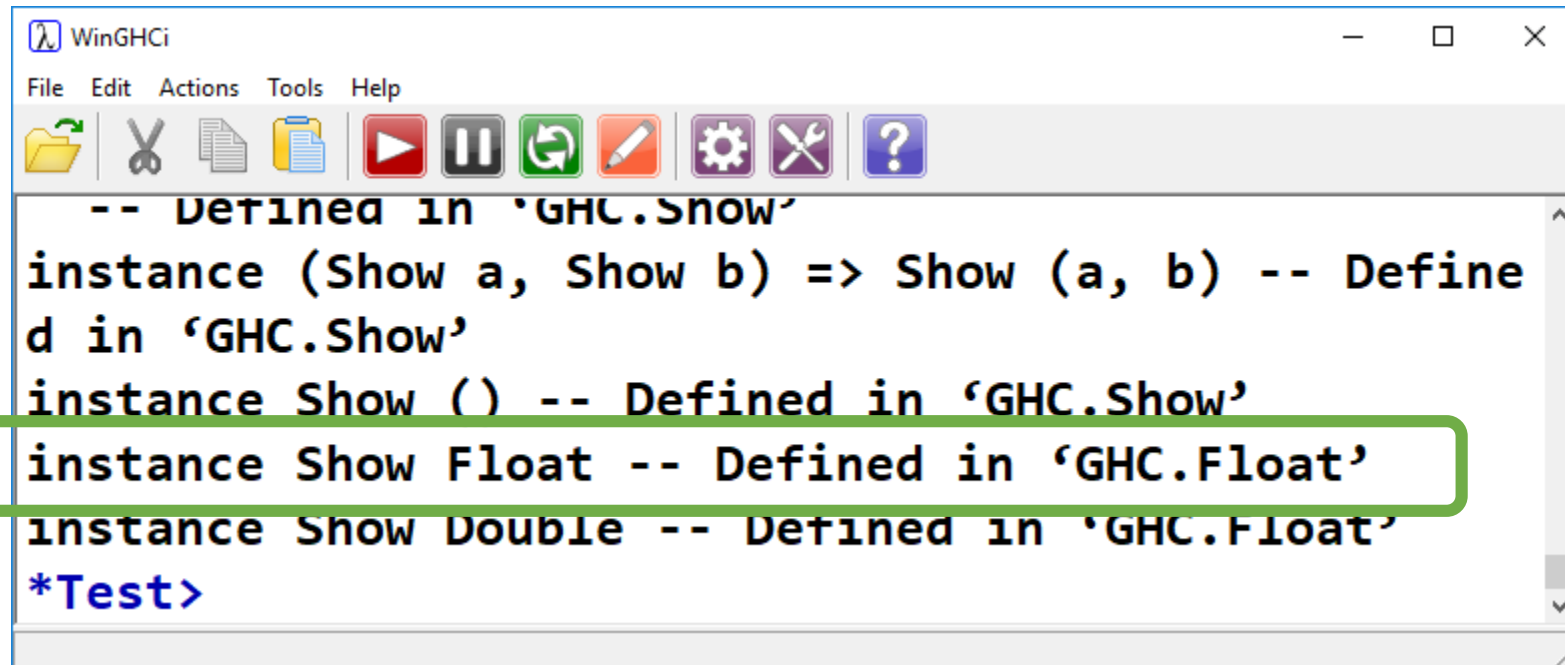
The output shows an error:

```
<interactive>:508:1: error:
  • No instance for (Show (Pt3 a0)) arising from
    interactive GHCi command
```

A red box highlights the text: "Echoing a value in the interactive window requires its type to be an instance of Show!".

Hmmm...

- The values contained in Pt3 are Float, and we know that Float is an instance of Show.
- How can we access the individual elements of Pt3?



```
-- Defined in 'GHC.Show'
instance (Show a, Show b) => Show (a, b) -- Defined
in 'GHC.Show'
instance Show () -- Defined in 'GHC.Show'
instance Show Float -- Defined in 'GHC.Float'
instance Show Double -- Defined in 'GHC.Float'
*Test>
```



```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt3 = Pt3 Float Float Float
4
5 ptX (Pt3 x y z) = x
6 ptY (Pt3 x y z) = y
7 ptZ (Pt3 x y z) = z
8
```

- Three access functions, one for each of the three values.
- Take as arguments Pt3 (and by extension its three members)
- Return x, y, or z coordinate respectively.

```
WinGHCi
File Edit Actions Tools Help
*Test> coord = Pt3 1 2 3
*Test> ptX coord
1.0
*Test> ptY coord
2.0
*Test> ptZ coord
3.0
*Test> |
```

Overloading Constructor

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5
6 ptX (Pt3 x y z) = x
7 ptY (Pt3 x y z) = y
8 ptZ (Pt3 x y z) = z
9
10
11
12
13
length: 1,358 lines: 78 Ln: 12 Col: 2 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

- Define Pt3 with three parameters
- Define Pt2 with two parameters
- Name of our data type is now simply Pt, because we have made it more generic.

There is now a problem with our access functions

There is now a problem with our access functions.

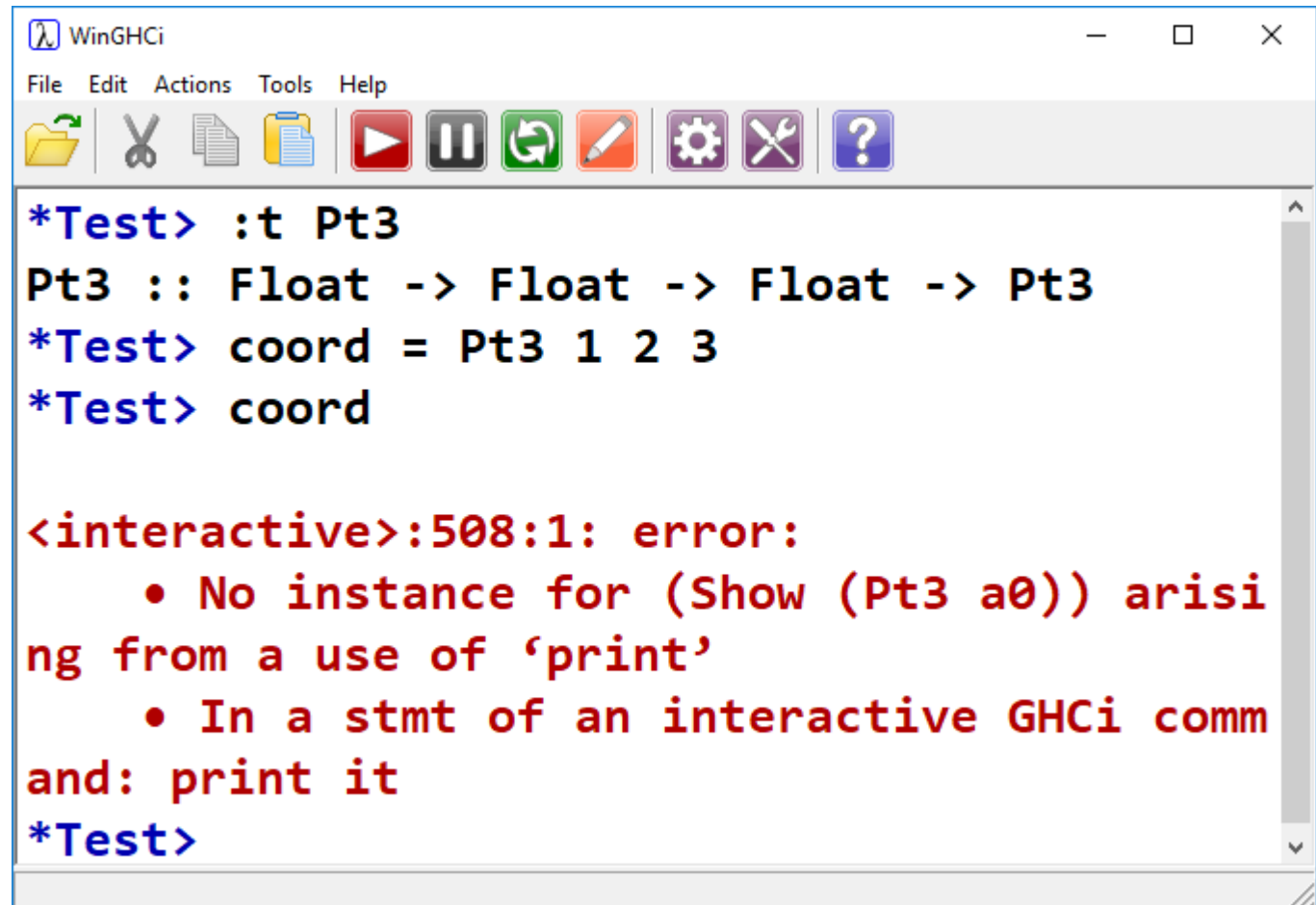
```
*C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5
6 ptX (Pt2 x _) = x
7 ptX (Pt3 x _ _) = x
8
9 ptY (Pt2 _ y) = y
10 ptY (Pt3 _ y _) = y
11
12 ptZ (Pt3 _ _ z) = z
13
14
length: 1,404 lines: 8 Ln: 14 Col: 2 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

Now our access functions work for both Pt2 and Pt3

```
WinGHCi
File Edit Actions Tools Help
*Test> coord2 = Pt2 3 4
*Test> coord3 = Pt3 5 6 7
*Test> ptX coord2
3.0
*Test> ptX coord3
5.0
*Test> ptY coord3
6.0
*Test> ptY coord2
4.0
*Test>
```

Deriving Show

Recall:



```
WinGHCi
File Edit Actions Tools Help
[Icons: Folder, Scissors, Document, Clipboard, Play, Pause, Refresh, Pencil, Gear, Wrench, Question Mark]

*Test> :t Pt3
Pt3 :: Float -> Float -> Float -> Pt3
*Test> coord = Pt3 1 2 3
*Test> coord

<interactive>:508:1: error:
    • No instance for (Show (Pt3 a0)) arising from a use of 'print'
    • In a stmt of an interactive GHCi command: print it
*Test>
```

Deriving Show

```
*C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7
11
12 ptY (Pt2 v) = v
length: 1,437 lines: 8 Ln: 8 Col: 2 Sel: 0|0 Windows (CR LF) UTF
```

Our custom type will inherit some default display behavior from **Show**

```
WinGHCi
File Edit Actions Tools Help
*Test> c2 = Pt2 1 2
*Test> c3 = Pt3 5 6 7
*Test> c2
Pt2 1.0 2.0
*Test> c3
Pt3 5.0 6.0 7.0
*Test> |
```

Similar to the toString() method in Java!

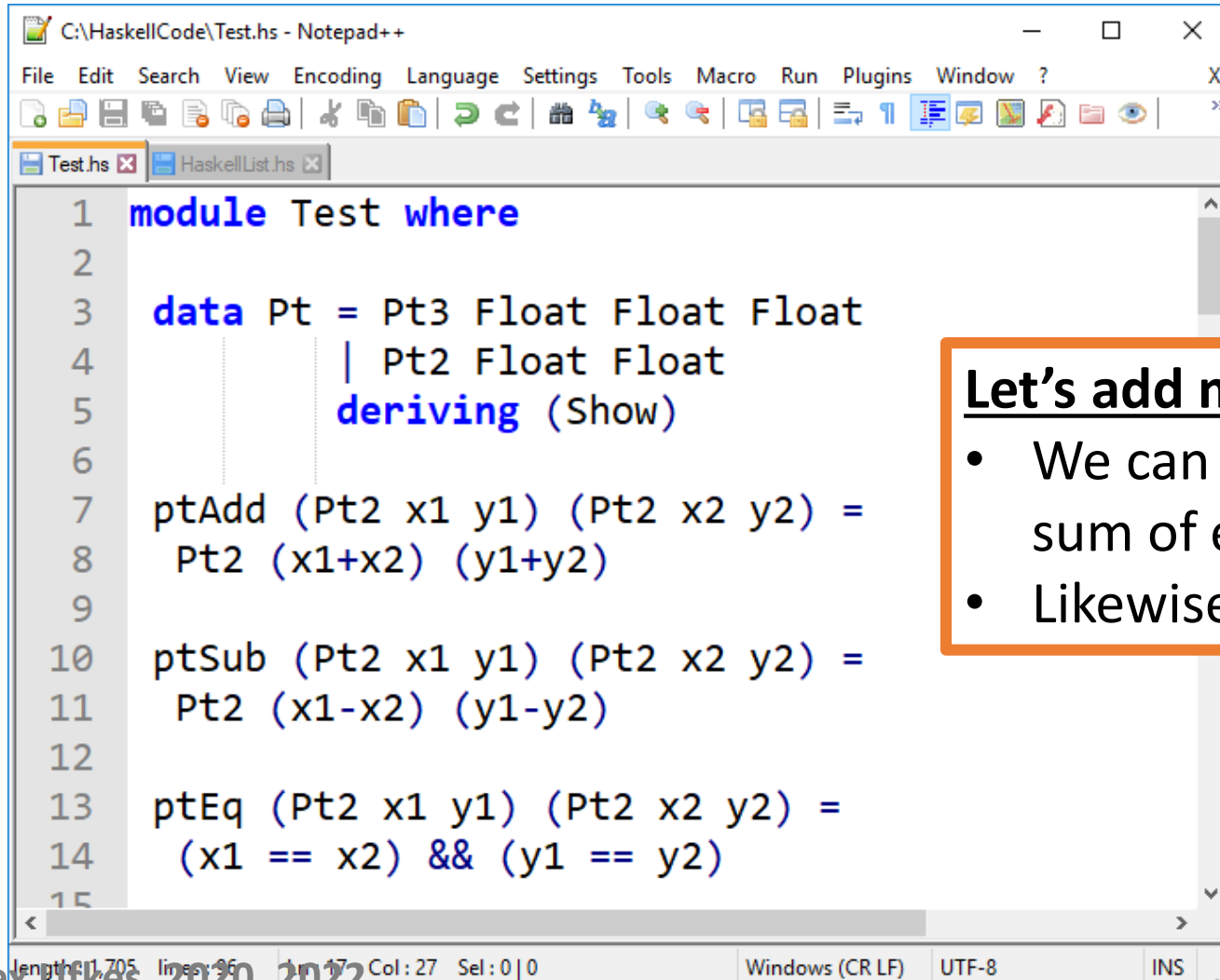
More Advanced Functions

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 vecLen (Pt2 x y) = sqrt(x^2 + y^2)
8 vecLen (Pt3 x y z) = sqrt(x^2 + y^2 + z^2)
9
10
11
12
length: 1,516 lines: 86 Ln: 14 Col: 21 Sel: 0 | 0 Windows (CR LF) UTF-8
```

Compute length of Pt2 and Pt3,
treating them as vectors

```
WinGHCi
File Edit Actions Tools Help
*Test> c2 = Pt2 1 2
*Test> c3 = Pt3 5 6 7
*Test> vecLen c2
2.236068
*Test> vecLen c3
10.488089
*Test> |
```

Addition, Subtraction, Equality?

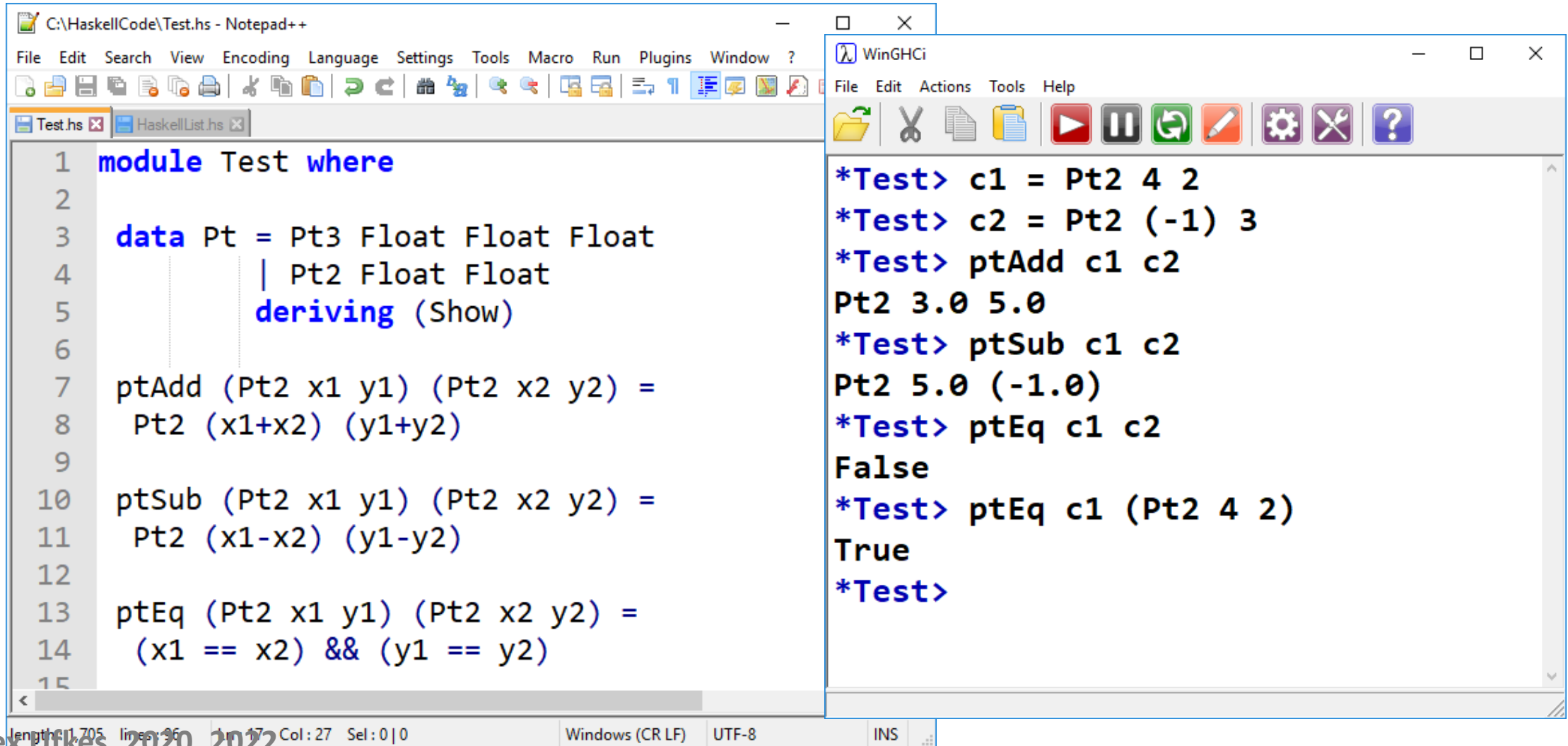


```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 ptAdd (Pt2 x1 y1) (Pt2 x2 y2) =
8     Pt2 (x1+x2) (y1+y2)
9
10 ptSub (Pt2 x1 y1) (Pt2 x2 y2) =
11     Pt2 (x1-x2) (y1-y2)
12
13 ptEq (Pt2 x1 y1) (Pt2 x2 y2) =
14     (x1 == x2) && (y1 == y2)
15
length: 1,705 lines: 96 Apr 17 Col: 27 Sel: 0|0 Windows (CR LF) UTF-8 INS
```

Let's add more functions!

- We can very easily define addition as the sum of each respective X and Y coord
- Likewise for subtraction and equality.

Addition, Subtraction, Equality?



The image shows a screenshot of a Haskell development environment. On the left is a Notepad++ window titled 'C:\HaskellCode\Test.hs - Notepad++' containing the following Haskell code:

```
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 ptAdd (Pt2 x1 y1) (Pt2 x2 y2) =
8     Pt2 (x1+x2) (y1+y2)
9
10 ptSub (Pt2 x1 y1) (Pt2 x2 y2) =
11     Pt2 (x1-x2) (y1-y2)
12
13 ptEq (Pt2 x1 y1) (Pt2 x2 y2) =
14     (x1 == x2) && (y1 == y2)
15
```

On the right is a WinGHCi window titled 'WinGHCi' showing the execution of the code:

```
*Test> c1 = Pt2 4 2
*Test> c2 = Pt2 (-1) 3
*Test> ptAdd c1 c2
Pt2 3.0 5.0
*Test> ptSub c1 c2
Pt2 5.0 (-1.0)
*Test> ptEq c1 c2
False
*Test> ptEq c1 (Pt2 4 2)
True
*Test>
```

The status bar at the bottom of the Notepad++ window shows: 'length: 1,705 lines: 96', 'Ln: 17 Col: 27 Sel: 0 | 0', 'Windows (CR LF)', 'UTF-8', and 'INS'.

Addition, Subtraction, Equality?

This seems very clunky. Why can't we simply add, subtract, or check equality with the symbolic operators (+, -, ==)?

We can! Equality is defined for instances of type class **Eq**
+, -, etc. are defined for instances of type class **Num**.

How do we make Pt2 and Pt3 instances of another type class?

Custom Types & Type Classes

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7
8
9 instance Eq Pt where
10     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
11
12
Haskell length: 1,623 lines: 99 Ln: 14 Col: 2 Sel: 0|0 Windows (CR LF) UTF-8 INS
```

Declare **Pt** to be an instance of **Eq**

Define what it means for two **Pt2** values to be considered equal

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Win
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7
8
9 instance Eq Pt where
10     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
11
12
Haskell length : 1,623 lines : 99 Ln : 14 Col : 2 Sel : 0 | 0 Windows (CR LF) UTF-8 INS
```

```
WinGHCi
File Edit Actions Tools Help
*Test> Pt2 1 2 == Pt2 2 3
False
*Test> Pt2 1 2 == Pt2 1 2
True
*Test>
```



Minimal Definition

The screenshot shows a WinGHCi window with a menu bar (File, Edit, Actions, Tools, Help) and a toolbar with icons for file operations and execution. The main area contains Haskell code:

```
*Test> :i Eq  
class Eq a where  
    (==) :: a -> a -> Bool  
    (/=) :: a -> a -> Bool  
    {-# MINIMAL (==) | (/=) #-}  
    -- Defined in 'GHC.Classes'  
  
instance [safe] Eq Pt -- Defined at Test.hs:7:11  
instance (Eq a) => Eq (List a) -- Defined at Test.hs:8:11  
instance Eq ()
```

An orange circle highlights the vertical bar in the minimal definition line. An orange-bordered box at the bottom right contains a bulleted explanation.

- The minimal definition for being an instance of **Eq** is == *OR* /= (not equal)
- We only defined ==

- The minimal definition for being an instance of **Eq** is `== *OR* /=` (not equal)
- We only defined `==`

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7
8 instance Eq Pt where
9     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
10
11
12
```

```
WinGHCi
File Edit Actions Tools Help
*Test> c1 = Pt2 2 3
*Test> c2 = Pt2 2 4
*Test> c1 == c2
False
*Test> c1 /= c2
True
*Test>
```

Haskell is clever enough to derive `/=` from our definition of `==`, and vice versa.

Let's Add /= Anyway

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7
8 instance Eq Pt where
9     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
10    (Pt2 x1 y1) /= (Pt2 x2 y2) = not (x1==x2 && y1==y2)
11
12
```

```
WinGHCi
File Edit Actions Tools Help
*Test> c1 = Pt2 2 3
*Test> c2 = Pt2 2 4
*Test> c1 == c2
False
*Test> c1 /= c2
True
*Test> |
```

Instance of Num

The image shows two windows from a Haskell development environment. The left window, titled 'C:\HaskellCode\Test.hs - Notepad++', displays the source code for a module named 'Test'. The code defines a data type 'Pt' with three constructors: 'Pt3', 'Pt2', and 'Pt1', each taking two 'Float' arguments. It also defines a 'Show' instance for 'Pt' using 'deriving'. An 'instance Num Pt where' block is partially visible, showing the start of a definition for '(Pt2 x1 y1) + (Pt2 x2 y2)'. The right window, titled 'WinGHCi', shows the GHCi prompt and the compilation of 'Test.hs'. It displays a warning: 'Test.hs:7:11: warning: [-Wmissing-methods]'. The warning lists two issues: 'No explicit implementation for' followed by a list of methods ('*', 'abs', 'signum', 'fromInteger', and 'negate') and 'In the instance declaration for 'Num Pt''. Below the warning, the GHCi prompt shows the successful loading of the module 'Test'.

```
1 module Test where
2
3 data Pt = Pt3 Float Float
4         | Pt2 Float Float
5         | Pt1 Float Float
6         deriving (Show)
7 instance Num Pt where
8     (Pt2 x1 y1) + (Pt2 x2 y2) = Pt2 (x1 + x2) (y1 + y2)
9
10
11
12
```

[1 of 1] Compiling Test (Test.hs, interpreted)

Test.hs:7:11: warning: [-Wmissing-methods]

- No explicit implementation for
 '*, 'abs', 'signum', 'fromInteger', and
 (either 'negate' or '-')
- In the instance declaration for 'Num Pt'

```
7 | instance Num Pt where
Ok, one module loaded.
*Test>
```

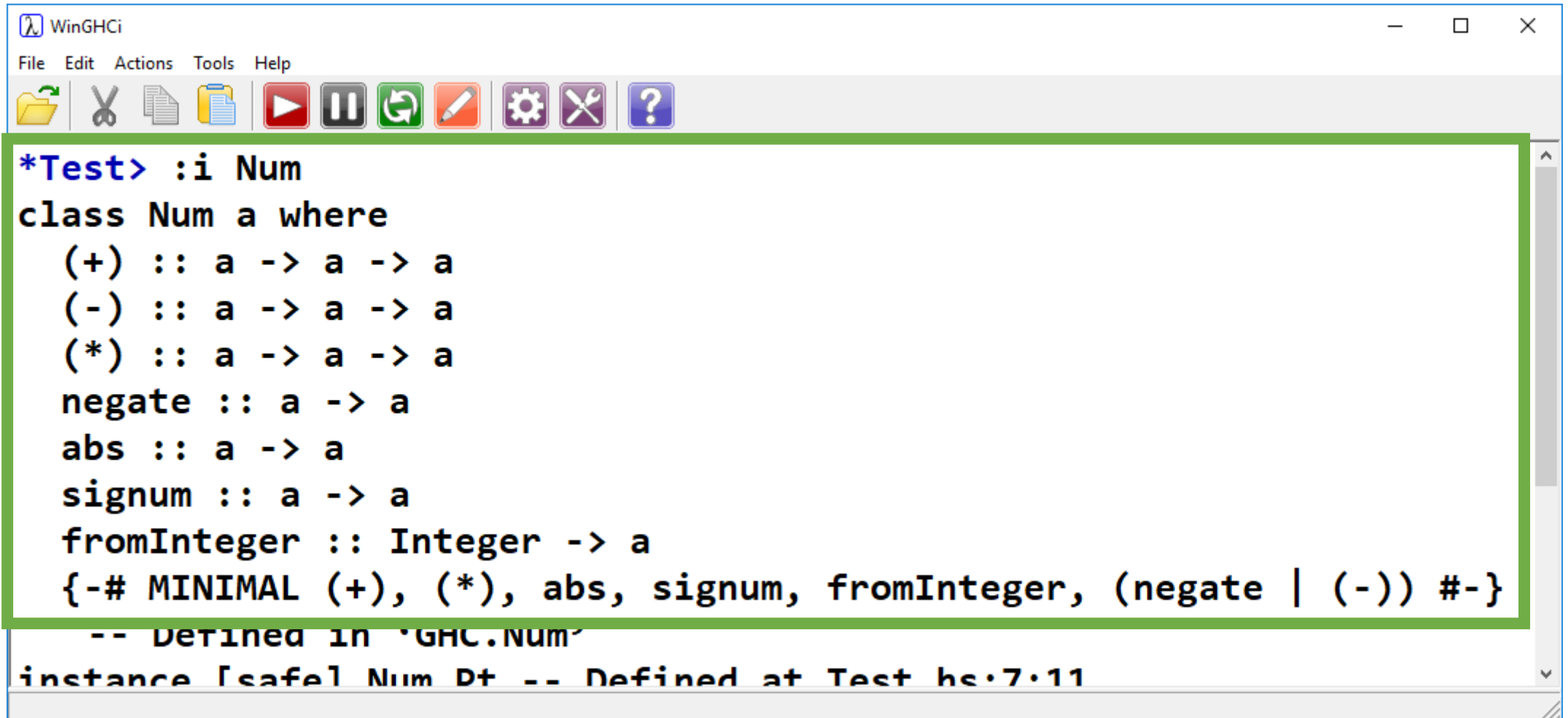
```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 instance Num Pt where
8     (Pt2 x1 y1) + (Pt2 x2 y2) = Pt2 (x1+x2) (y1+y2)
9
10
11
12
```

- We're only implementing for Pt2.
- Adding Pt3 follows the same pattern

```
WinGHCi
File Edit Actions Tools Help
*Test> Pt2 1 2 + Pt2 3 4
Pt2 4.0 6.0
*Test> x = Pt2 1 2
*Test> y = Pt2 6 7
*Test> x+y
Pt2 7.0 9.0
*Test>
```

```
WinGHCi
File Edit Actions Tools Help
*Test> Pt3 1 2 3 + Pt3 1 2 3
*** Exception: Test.hs:8:3-49: Non-exhaustive patterns in function +
*Test> |
```


Instance of Num

A screenshot of a WinGHCi window. The window has a title bar with the text 'WinGHCi' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Under the menu bar is a toolbar with icons for file operations (folder, scissors, document), execution (play, pause, refresh), editing (pencil), settings (gear, wrench), and help (question mark). The main text area contains the following Haskell code:

```
*Test> :i Num
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)) #-}
  -- Defined in 'GHC.Num'
instance [safe] Num D+ -- Defined at Test.hs:7:11
```

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 instance Num Pt where
8     (Pt2 x1 y1) + (Pt2 x2 y2) = Pt2 (x1 + x2) (y1 + y2)
9     (Pt2 x1 y1) - (Pt2 x2 y2) = Pt2 (x1 - x2) (y1 - y2)
10    (Pt2 x1 y1) * (Pt2 x2 y2) = Pt2 (x1 * x2) (y1 * y2)
11    abs (Pt2 x1 y1) = Pt2 (abs x1) (abs y1)
12    signum (Pt2 x1 y1) = Pt2 (signum x1) (signum y1)
13
14 instance Eq Pt where
15     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
16     (Pt2 x1 y1) /= (Pt2 x2 y2) = not (x1==x2 && y1==y2)
17
18
```

- This may look circular
- We're using abs and signum in our definition of abs and signum.
- However! x1 and y1 are Float.
- abs and signum *are* defined for Float
- We're defining them for Pt2

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 instance Num Pt where
8     (Pt2 x1 y1) + (Pt2 x2 y2) = Pt2 (x1+x2) (y1+y2)
9     (Pt2 x1 y1) - (Pt2 x2 y2) = Pt2 (x1-x2) (y1-y2)
10    (Pt2 x1 y1) * (Pt2 x2 y2) = Pt2 (x1*x2) (y1*y2)
11    abs (Pt2 x1 y1) = Pt2 (abs x1) (abs y1)
12    signum (Pt2 x1 y1) = Pt2 (signum x1) (signum y1)
13    fromInteger n = let a = (fromInteger n) in Pt2 a a
14
15 instance Eq Pt where
16     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
17     (Pt2 x1 y1) /= (Pt2 x2 y2) = not (x1==x2 && y1==y2)
18
Haskell length: 2,004 lines: 105 Ln: 21 Col: 2 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

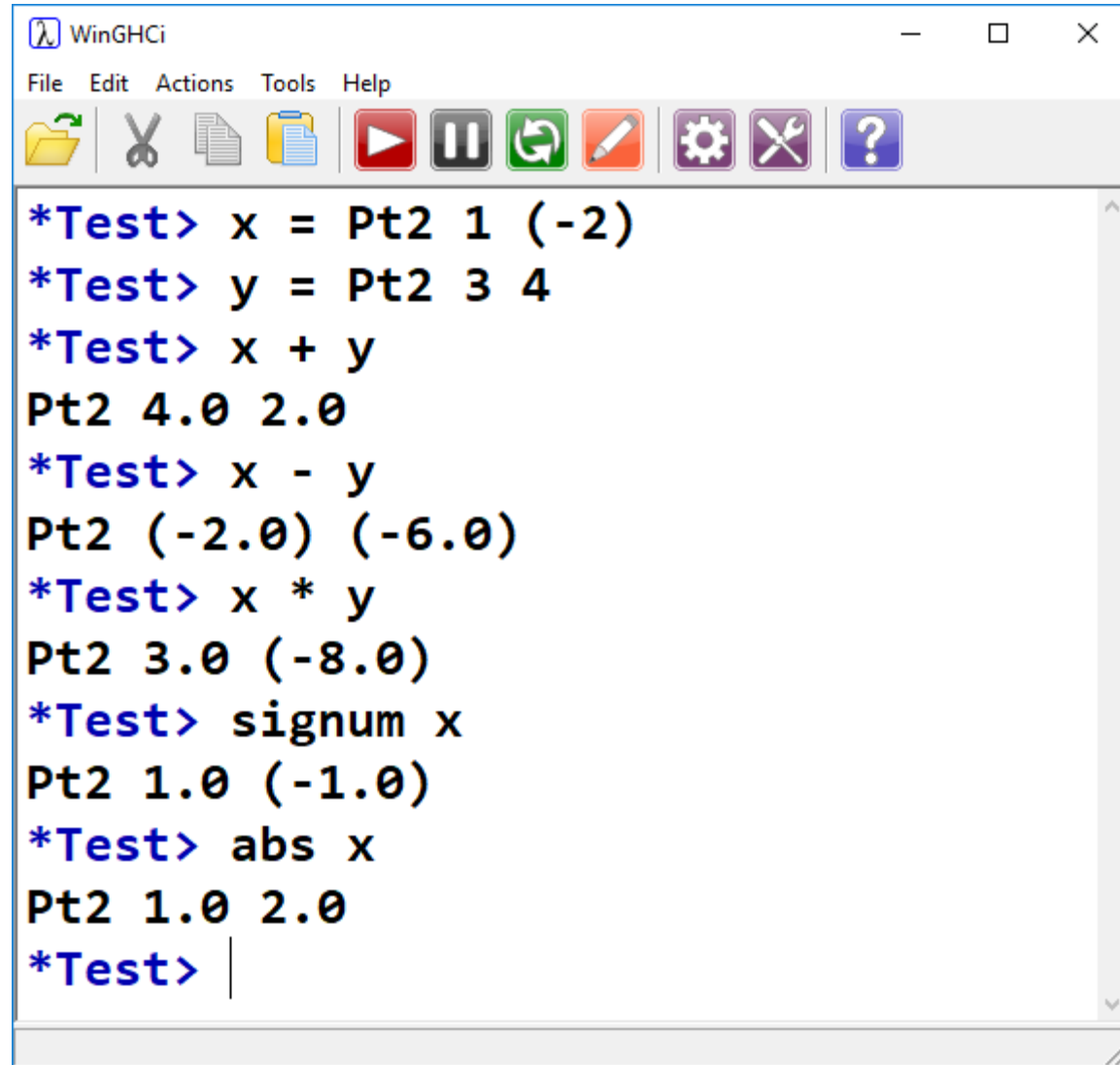
- fromInteger is a *coercion* function.
- Dictates how our custom type can be created from an Integer
- Takes an Integer, returns a Pt
- Allows us to do this...

```
WinGHCi
File Edit Actions Tools Help
*Test> (Pt2 2 3) + 4
Pt2 6.0 7.0
*Test>
```

```
C:\HaskellCode\Test.hs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Test.hs HaskellList.hs
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         deriving (Show)
6
7 instance Num Pt where
8     (Pt2 x1 y1) + (Pt2 x2 y2) = Pt2 (x1+x2) (y1+y2)
9     (Pt2 x1 y1) - (Pt2 x2 y2) = Pt2 (x1-x2) (y1-y2)
10    (Pt2 x1 y1) * (Pt2 x2 y2) = Pt2 (x1*x2) (y1*y2)
11    abs (Pt2 x1 y1) = Pt2 (abs x1) (abs y1)
12    signum (Pt2 x1 y1) = Pt2 (signum x1) (signum y1)
13    fromInteger n = let a = (fromInteger n) in Pt2 a a
14
15 instance Eq Pt where
16     (Pt2 x1 y1) == (Pt2 x2 y2) = (x1==x2 && y1==y2)
17     (Pt2 x1 y1) /= (Pt2 x2 y2) = not (x1==x2 && y1==y2)
18
Haskell length: 2,004 lines: 105 Ln: 21 Col: 2 Sel: 0|0 Windows (CR LF) UTF-8 INS
```

```
WinGHCi
File Edit Actions Tools Help
*Test> :reload
[1 of 1] Compiling Test
( Test.hs, interpreted )
Ok, one module loaded.
*Test> |
```

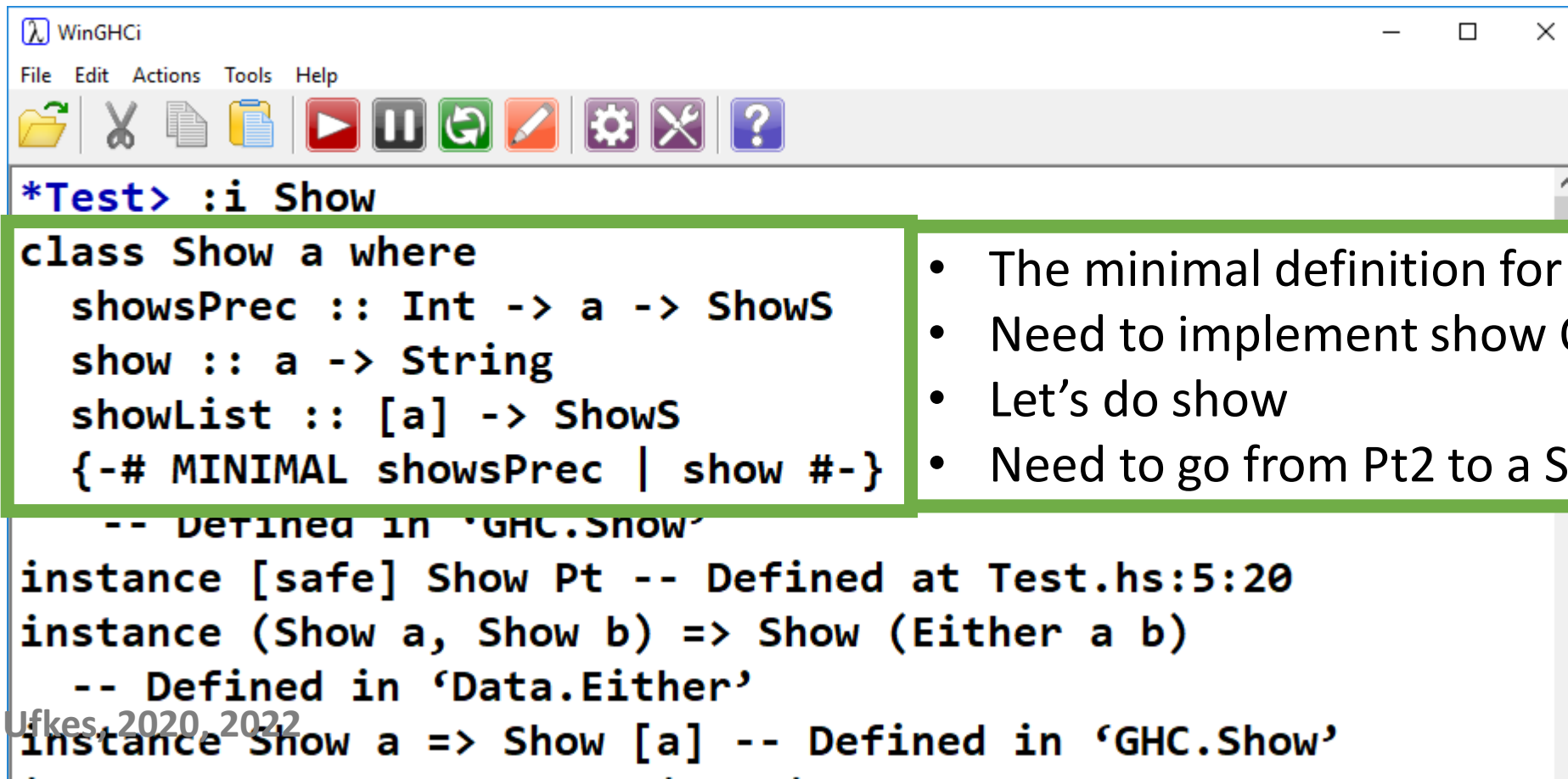
No more warnings!

A screenshot of the WinGHCi window. The window has a title bar with the WinGHCi logo and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Under the 'Actions' menu, there is a toolbar with icons for file operations (copy, paste, save), execution (run, pause, step over, step into), and settings (gear, wrench, help). The main text area contains the following Haskell code and its output:

```
*Test> x = Pt2 1 (-2)
*Test> y = Pt2 3 4
*Test> x + y
Pt2 4.0 2.0
*Test> x - y
Pt2 (-2.0) (-6.0)
*Test> x * y
Pt2 3.0 (-8.0)
*Test> signum x
Pt2 1.0 (-1.0)
*Test> abs x
Pt2 1.0 2.0
*Test> |
```

Instance of Show

In Java-speak, define our own toString(), instead of deriving the default

A screenshot of the WinGHCi Haskell interpreter window. The window has a title bar 'WinGHCi' and a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for file operations (open, save, copy, paste), execution (run, stop, refresh), and settings (preferences, help). The main text area shows the following content:

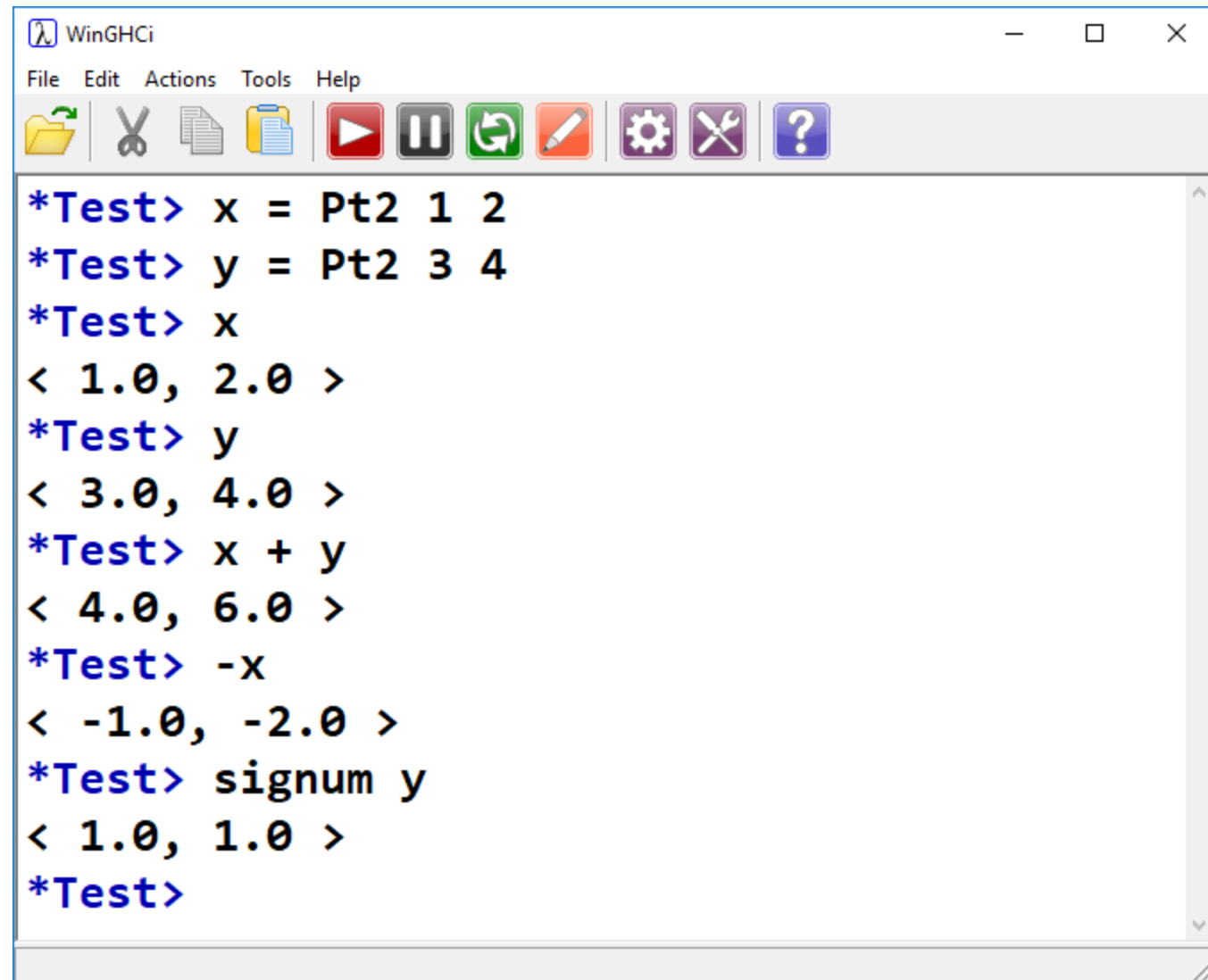
```
*Test> :i Show
class Show a where
  showsPrec :: Int -> a -> ShowS
  show :: a -> String
  showList :: [a] -> ShowS
  {-# MINIMAL showsPrec | show #-}
  -- Defined in 'GHC.Show'
instance [safe] Show Pt -- Defined at Test.hs:5:20
instance (Show a, Show b) => Show (Either a b)
  -- Defined in 'Data.Either'
instance Show a => Show [a] -- Defined in 'GHC.Show'
```

- The minimal definition for Show is easy
- Need to implement show OR showsPrec
- Let's do show
- Need to go from Pt2 to a String

```
1 module Test where
2
3 data Pt = Pt3 Float Float Float
4         | Pt2 Float Float
5         --deriving (Show)
6
7 instance Show Pt where
8     show (Pt2 x y) =
9         "< " ++ (show x) ++ ", " ++ (show y) ++ " >"
10
11
```

No longer need to derive Show, we've made our own

- Use string concatenation to create a pleasing visual output for Pt2
- In doing so, we make use of show as defined for Floats

A screenshot of the WinGHCi window. The window has a title bar with the WinGHCi logo and standard window controls. Below the title bar is a menu bar with 'File', 'Edit', 'Actions', 'Tools', and 'Help'. Under the 'Actions' menu, there is a toolbar with icons for file operations (open, save, copy, paste), execution (run, pause, step over, step into), and help (settings, close, question mark). The main text area contains the following Haskell code and its output:

```
*Test> x = Pt2 1 2
*Test> y = Pt2 3 4
*Test> x
< 1.0, 2.0 >
*Test> y
< 3.0, 4.0 >
*Test> x + y
< 4.0, 6.0 >
*Test> -x
< -1.0, -2.0 >
*Test> signum y
< 1.0, 1.0 >
*Test>
```


Haskell Tutorials/References:

https://en.wikibooks.org/wiki/Yet_Another_Haskell_Tutorial

<http://cheatsheet.codeslower.com/CheatSheet.pdf>

