FUNCTIONAL PROGRAMMING 2016

QUESTION 3

```
> data Zig a b = Nil | Cins a (Zag b a) deriving Show
> data Zag a b = Nal | Cams a (Zig b a) deriving Show

> exampleZig :: Zig Integer Char
> exampleZig = Cins 1 (Cams 'A' (Cins 2 Nal))

> exampleZag :: Zag String Bool
> exampleZag = Cams "C" (Cins True (Cams "D" Nil))
```

(a)

```
> headZig :: Zig a b -> a
> headZig (Cins x y) = x

> headZag :: Zag a b -> a
> headZag (Cams x y) = x
```

(b)

```
> data ZigOrZagType a b = ZigType a | ZagType b deriving Show
> lastZig :: Zig a b -> ZigOrZagType a b
> lastZig (Cins x Nal) = ZigType x
> lastZig (Cins x zag) = lastZag zag

> lastZag :: Zag a b -> ZigOrZagType a b
> lastZag (Cams x Nil) = ZagType x
> lastZag (Cams x zig) = lastZig zig
```

(c)

```
> lastZig (Cins ⊥ (Cams 'A' Nil))
zagType 'A'
```

This happens because (Cams 'A' Nil) is not Nal, therefore, from the pattern-matching of the lastZig function, we will get to calculate lastZag of (Cams 'A' Nil), which is zagType 'A'.

```
> lastZig (Cins ⊥ Nal)
ZigType
```

The result will be ZigType ⊥ , from pattern-matching, thus, after ZigType is printed, the Command Prompt freezes trying to print ⊥.

```
> lastZig (Cins ⊥ Nil)
```

We will get an error because, from pattern-matching, the program will try to calculate lastZag Nil, but Nil is of type Zig a b, which comes into contradiction with the definition of last Zag.

(d)

```
> mapZig :: (a -> a) -> (b -> b) -> Zig a b -> Zig a b
> mapZig f g Nil = Nil
> mapZig f g (Cins x zag) = Cins (f x) (mapZag g f zag)

> mapZag :: (b -> b) -> (a -> a) -> Zag b a -> Zag b a
> mapZag g f Nal = Nal
> mapZag g f (Cans y zig) = Cans (g y) (mapZig f g zig)
```

(e)

```
> foldZig :: ((x -> c -> b), b) -> ((y -> b -> c), c) -> Zig x y -> b
> foldZig (fZig, eZig) (fZag, eZag) Nil = eZig
> foldZig (fZig, eZig) (fZag, eZag) (Cins x z) =
>         fZig x (foldZag (fZag, eZag) (fZig, eZig) z)

> foldZag :: ((y -> b -> c), c) -> ((x -> c -> b), b) -> Zag y x -> c
> foldZag (fZag, eZag) (fZig, eZig) Nal = eZag
> foldZag (fZag, eZag) (fZig, eZig) (Cans y z) =
>         fZag y (foldZig (fZig, eZig) (fZag, eZag) z)
```

To determine the type signatures, we start from the types of eZig :: b and eZag :: c. In the first cases of the pattern-matching, we can see that the result of foldZig is of type b and thus the result of fZig is of type b (second case of the pattern-matching). Same goes for foldZag and fZag.

So, fZig :: x -> c -> b and fZag :: y -> b -> c. From these, we can easily deduce the type signatures for foldZig and foldZag.