# Agda

March 7, 2013

```
module helloworld where

open import IO.Primitive using (IO; putStrLn)
open import Data.String using (toCostring; String)
open import Foreign.Haskell using (Unit)
open import Function

main : IO Unit
main = putStrLn (toCostring "Hello, world!")
```

Task: Provide a proven correct implementation of lists.

Requirement: Must be able to give checked access to the first element of the list.

```
module safeList where

data Nat : Set where
  zero : Nat
  suc  : Nat -> Nat
```

```
module safeList where

data Nat : Set where
  zero : Nat
  suc  : Nat -> Nat

_+_ : Nat -> Nat -> Nat
zero + zero = zero
zero + n = n
(suc n1) + n2 = suc (n1 + n2)
```

```
module safeList where

data Nat : Set where
  zero : Nat
  suc  : Nat -> Nat

_+_ : Nat -> Nat -> Nat
zero + zero = zero
zero + n = n
(suc n1) + n2 = suc (n1 + n2)

data List (a : Set) : Nat -> Set where
  [] : List a zero
  _::_ : {n : Nat} -> a -> List a n -> List a (suc n)
```

```
module safeList where

data Nat : Set where
  zero : Nat
  suc  : Nat -> Nat

_+_ : Nat -> Nat -> Nat
zero + zero = zero
zero + n = n
(suc n1) + n2 = suc (n1 + n2)

data List (a : Set) : Nat -> Set where
  [] : List a zero
  _::_ : {n : Nat} -> a -> List a n -> List a (suc n)

head : {A : Set}{n : Nat} -> List A (suc n) -> A
head (x :: xs) = x
```