Prelude

In this appendix the entire Haskell Prelude is given. It constitutes a specification for the Prelude. Many of the definitions are written with clarity rather than efficiency in mind, and it is not required that the specification be implemented as shown here.

The Prelude shown here is organized into a root module, Prelude, and three sub-modules, PreludeList, PreludeText, and PreludeIO. This structure is purely presentational. An implementation is not required to use this organisation for the Prelude, nor are these three modules available for import separately. Only the exports of module Prelude are significant.

Some of these modules import Library modules, such as Char, Monad, IO, and Numeric. These modules are described fully in the accompanying Haskell 98 Library Report. These imports are not, of course, part of the specification of the Prelude. That is, an implementation is free to import more, or less, of the Library modules, as it pleases.

Primitives that are not definable in Haskell , indicated by names starting with "prim", are defined in a system dependent manner in module PreludeBuiltin and are not shown here. Instance declarations that simply bind primitives to class methods are omitted. Some of the more verbose instances with obvious functionality have been left out for the sake of brevity.

Declarations for special types such as Integer, (), or (->) are included in the Prelude for completeness even though the declaration may be incomplete or syntactically invalid.

```
module Prelude (
    module PreludeList, module PreludeText, module PreludeIO,
    Bool(False, True),
    Maybe(Nothing, Just),
    Either(Left, Right),
    Ordering(LT, EQ, GT),
    Char, String, Int, Integer, Float, Double, Rational, IO,
--  List type: [](:), [])
--  Tuple types: (,), (,,), etc.
--  Trivial type: ()
--  Functions: (->)
    Eq((==), (/=)),
    Ord(compare, (<), (<=), (>=), (>), max, min),
    Enum(succ, pred, toEnum, fromEnum, enumFrom, enumFromThen,
         enumFromTo, enumFromThenTo),
    Bounded(minBound, maxBound),
    Num((+), (-), (*), negate, abs, signum, fromInteger),
    Real(toRational),
    Integral(quot, rem, div, mod, quotRem, divMod, toInteger),
    Fractional((/), recip, fromRational),
    Floating(pi, exp, log, sqrt, (**), logBase, sin, cos, tan,
             asin, acos, atan, sinh, cosh, tanh, asinh, acosh, atanh),
    RealFrac(properFraction, truncate, round, ceiling, floor),
    RealFloat(floatRadix, floatDigits, floatRange, decodeFloat,
              encodeFloat, exponent, significand, scaleFloat, isNaN,
              isInfinite, isDenormalized, isIEEE, isNegativeZero, atan2),
    Monad((>>=), (>>), return, fail),
```

```
    Functor(fmap),
    mapM, mapM_, sequence, sequence_, (=<<),
    maybe, either,
    (&&), (||), not, otherwise,
    subtract, even, odd, gcd, lcm, (^), (^^),
    fromIntegral, realToFrac,
    fst, snd, curry, uncurry, id, const, (.), flip, ($), until,
    asTypeOf, error, undefined,
    seq, ($!)
  ) where

import PreludeBuiltin  -- Contains all `prim' values
import PreludeList
import PreludeText
import PreludeIO
import Ratio( Rational )

infixr 9  .
infixr 8  ^, ^^, **
infixl 7  *, /, `quot`, `rem`, `div`, `mod`
infixl 6  +, -
infixr 5  :
infix  4  ==, /=, <, <=, >=, >
infixr 3  &&
infixr 2  ||
infixl 1  >>, >>=
infixr 1  =<<
infixr 0  $, $!, `seq`


module PreludeList (
    map, (++), filter, concat,
    head, last, tail, init, null, length, (!!),
    foldl, foldl1, scanl, scanl1, foldr, foldr1, scanr, scanr1,
    iterate, repeat, replicate, cycle,
    take, drop, splitAt, takeWhile, dropWhile, span, break,
    lines, words, unlines, unwords, reverse, and, or,
    any, all, elem, notElem, lookup,
    Sum, product, maximum, minimum, concatMap,
    zip, zip3, zipWith, zipWith3, unzip, unzip3)
  where

import qualified Char(isSpace)

infixl 9  !!
infixr 5  ++
infix  4  `elem`, `notElem`


module PreludeText (
    ReadS, ShowS,
    Read(readsPrec, readList),
    Show(showsPrec, showList),
    reads, shows, show, read, lex,
    showChar, showString, readParen, showParen ) where


-- The instances of Read and Show for
-- Bool, Char, Maybe, Either, Ordering
-- are done via "deriving" clauses in Prelude.hs

import Char(isSpace, isAlpha, isDigit, isAlphaNum,
            showLitChar, readLitChar, lexLitChar)

import Numeric(showSigned, showInt, readSigned, readDec, showFloat,
               readFloat, lexDigits)


module PreludeIO (
```

```
      FilePath, IOError, ioError, userError, catch,
      putChar, putStr, putStrLn, print,
      getChar, getLine, getContents, interact,
      readFile, writeFile, appendFile, readIO, readLn
    ) where

  import PreludeBuiltin
```