JFP **13** (1): 153–156, January 2003.
DOI: 10.1017/S0956796803001515 Printed in the United Kingdom

Chapter 13

Complex Numbers

```
module  Complex (
    Complex((:+)), realPart, imagPart, conjugate,
    mkPolar, cis, polar, magnitude, phase ) where

infix  6  :+

data  (RealFloat a)     => Complex a = !a :+ !a

realPart, imagPart      :: (RealFloat a) => Complex a -> a
conjugate               :: (RealFloat a) => Complex a -> Complex a
mkPolar                 :: (RealFloat a) => a -> a -> Complex a
cis                     :: (RealFloat a) => a -> Complex a
polar                   :: (RealFloat a) => Complex a -> (a,a)
magnitude, phase        :: (RealFloat a) => Complex a -> a

instance  (RealFloat a) => Eq         (Complex a)  where ...
instance  (RealFloat a) => Read       (Complex a)  where ...
instance  (RealFloat a) => Show       (Complex a)  where ...
instance  (RealFloat a) => Num        (Complex a)  where ...
instance  (RealFloat a) => Fractional (Complex a)  where ...
instance  (RealFloat a) => Floating   (Complex a)  where ...
```

Complex numbers are an algebraic type. The constructor `(:+)` forms a complex number from its real and imaginary rectangular components. This constructor is strict: if either the real part or the imaginary part of the number is $\perp$, the entire number is $\perp$. A complex number may also be formed from polar components of magnitude and phase by the function `mkPolar`. The function

`cis` produces a complex number from an angle t. Put another way, `cis` t is a complex value with magnitude 1 and phase t (modulo 2π).

The function `polar` takes a complex number and returns a (magnitude, phase) pair in canonical form: The magnitude is nonnegative, and the phase, in the range $(-\pi, \pi]$; if the magnitude is zero, then so is the phase.

The functions `realPart` and `imagPart` extract the rectangular components of a complex number and the functions `magnitude` and `phase` extract the polar components of a complex number. The function `conjugate` computes the conjugate of a complex number in the usual way.

The magnitude and sign of a complex number are defined as follows:

```
abs z              =  magnitude z :+ 0
signum 0           =  0
signum z@(x:+y)    =  x/r :+ y/r  where r = magnitude z
```

That is, `abs` z is a number with the magnitude of z, but oriented in the positive real direction, whereas `signum` z has the phase of z, but unit magnitude.

13.1 Library `Complex`

```
module Complex(Complex((:+)), realPart, imagPart, conjugate, mkPolar,
               cis, polar, magnitude, phase)  where

infix  6  :+

data  (RealFloat a)     => Complex a = !a :+ !a  deriving (Eq,Read,Show)

realPart, imagPart :: (RealFloat a) => Complex a -> a
realPart (x:+y)  =  x
imagPart (x:+y)  =  y

conjugate        :: (RealFloat a) => Complex a -> Complex a
conjugate (x:+y) =  x :+ (-y)

mkPolar          :: (RealFloat a) => a -> a -> Complex a
mkPolar r theta  =  r * cos theta :+ r * sin theta

cis              :: (RealFloat a) => a -> Complex a
cis theta        =  cos theta :+ sin theta

polar            :: (RealFloat a) => Complex a -> (a,a)
polar z          =  (magnitude z, phase z)

magnitude :: (RealFloat a) => Complex a -> a
magnitude (x:+y) =  scaleFloat k
                     (sqrt ((scaleFloat mk x)^2 + (scaleFloat mk y)^2))
                    where k  = max (exponent x) (exponent y)
                          mk = - k
```

https://doi.org/10.1017/S0956796803001515 Published online by Cambridge University Press

```
phase :: (RealFloat a) => Complex a -> a
phase (0 :+ 0) = 0
phase (x :+ y) = atan2 y x

instance  (RealFloat a) => Num (Complex a)  where
    (x:+y) + (x':+y')   =  (x+x') :+ (y+y')
    (x:+y) - (x':+y')   =  (x-x') :+ (y-y')
    (x:+y) * (x':+y')   =  (x*x'-y*y') :+ (x*y'+y*x')
    negate (x:+y)       =  negate x :+ negate y
    abs z               =  magnitude z :+ 0
    signum 0            =  0
    signum z@(x:+y)     =  x/r :+ y/r  where r = magnitude z
    fromInteger n       =  fromInteger n :+ 0

instance  (RealFloat a) => Fractional (Complex a)  where
    (x:+y) / (x':+y')   =  (x*x''+y*y'') / d :+ (y*x''-x*y'') / d
                           where x'' = scaleFloat k x'
                                 y'' = scaleFloat k y'
                                 k   = - max (exponent x') (exponent y')
                                 d   = x'*x'' + y'*y''
    fromRational a      =  fromRational a :+ 0
```

https://doi.org/10.1017/S0956796803001515 Published online by Cambridge University Press

```
instance  (RealFloat a) => Floating (Complex a) where
    pi             =  pi :+ 0
    exp (x:+y)     =  expx * cos y :+ expx * sin y
                      where expx = exp x
    log z          =  log (magnitude z) :+ phase z
    sqrt 0         =  0
    sqrt z@(x:+y)  =  u :+ (if y < 0 then -v else v)
                      where (u,v) = if x < 0 then (v',u') else (u',v')
                            v'    = abs y / (u'*2)
                            u'    = sqrt ((magnitude z + abs x) / 2)
    sin (x:+y)     =  sin x * cosh y :+ cos x * sinh y
    cos (x:+y)     =  cos x * cosh y :+ (- sin x * sinh y)
    tan (x:+y)     =  (sinx*coshy:+cosx*sinhy)/(cosx*coshy:+(-sinx*sinhy))
                      where sinx  = sin x
                            cosx  = cos x
                            sinhy = sinh y
                            coshy = cosh y
    sinh (x:+y)    =  cos y * sinh x :+ sin  y * cosh x
    cosh (x:+y)    =  cos y * cosh x :+ sin y * sinh x
    tanh (x:+y)    =  (cosy*sinhx:+siny*coshx)/(cosy*coshx:+siny*sinhx)
                      where siny  = sin y
                            cosy  = cos y
                            sinhx = sinh x
                            coshx = cosh x
    asin z@(x:+y)  =  y':+(-x')
                      where  (x':+y') = log (((-y):+x) + sqrt (1 - z*z))
    acos z@(x:+y)  =  y'':+(-x'')
                      where (x'':+y'') = log (z + ((-y'):+x'))
                            (x':+y')   = sqrt (1 - z*z)
    atan z@(x:+y)  =  y':+(-x')
                      where (x':+y') = log (((1-y):+x) / sqrt (1+z*z))
    asinh z        =  log (z + sqrt (1+z*z))
    acosh z        =  log (z + (z+1) * sqrt ((z-1)/(z+1)))
    atanh z        =  log ((1+z) / sqrt (1-z*z))
```

https://doi.org/10.1017/S0956796803001515 Published online by Cambridge University Press