Chapter 15
Complex

Make it as simple as possible, but no simpler.
Albert Einstein

Aims
The aims of this chapter are:

- To introduce the last predefined numeric data type in Fortran.
- To illustrate with examples how to use this type.

15.1 Introduction

This variable type reflects an extension of the real data type available in Fortran — the complex data type, where we can store and manipulate complex variables. Problems that require this data type are restricted to certain branches of mathematics, physics and engineering. Complex numbers are defined as having a real and imaginary part, i.e., $a = x + iy$ where i is the square root of –1.

They are not supported in many programming languages as a base type which makes Fortran the language of first choice for many people.

To use this variable type we have to write the number as two parts, the real and imaginary elements of the number, for example,

```
complex :: u
u=cmplx(1.0,2.0)
```

represents the complex number $1 + i2$. Note that the complex number is enclosed in brackets. We can do arithmetic on variables like this, and most of the intrinsic functions such as `log`, `sin`, `cos`, etc., accept a complex data type as argument.

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_15

All the usual rules about mixing different variable types, like reals and integers, also apply to complex. Complex numbers are read in and written out in a similar way to real numbers, but with the provision that, for each single complex value, two format descriptors must be given. You may use either E or F formats (or indeed, mix them), as long as there are enough of them. Although you use brackets around the pairs of numbers in a program, these must not appear in any input, nor will they appear on the output.

15.2 Example 1: Use of `cmplx`, `aimag` and `conjg`

There are a number of intrinsic functions to enable complex calculations to be performed. The program below uses some of them:

```
program ch1501
  implicit none
  complex :: z, z1, z2, z3, zbar
  real :: x, y, zmod
  real :: x2 = 3.0, y2 = 4.0
  real :: x3 = -2.0, y3 = -3.0

  z1 = cmplx(1.0, 2.0) !         1 + i 2
  z2 = cmplx(x2, y2)   !         x2 + i y2
  z3 = cmplx(x3, y3)   !         x3 + i y3
  z = z1*z2/z3
  x = real(z)          !         real part of
!                                z
  y = aimag(z)         !         imaginary
!                                part of z
  zmod = abs(z)        !         modulus of z
  zbar = conjg(z)      !         complex
!                                conjugate of
!                                z
  print 100, z1, z2, z3
100 format (3(1x,f4.1,' + i ',f4.1,/))
  print 110, z, zmod, zbar
110 format (1x, f4.1, ' + i ', f4.1, /, 1x, &
    f4.1, /, 1x, f4.1, ' + i ', f4.1)
  print 120, x, y
120 format (2(1x,f4.1,/)) end program ch1501
```

15.3 Example 2: Polar Coordinate Example

The second order differential equation:

$$\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + y = x(t)$$

could describe the behaviour of an electrical system, where $x(t)$ is the input voltage and $y(t)$ is the output voltage and dy/dt is the current. The complex ratio

$$\frac{y(w)}{x(w)} = 1/(-w^2 + 2jw + 1)$$

is called the frequency response of the system because it describes the relationship between input and output for sinusoidal excitation at a frequency of w and where j is $\sqrt{(}-1)$ The following program reads in a value of w and evaluates the frequency response for this value of w together with its polar form (magnitude and phase):

```
program ch1502
  implicit none

! program to calculate frequency
! response of a system
! for a given omega
! and its polar form (magnitude and phase).

  real :: omega, real_part, imag_part, &
    magnitude, phase
  complex :: frequency_response

! Input frequency omega

  print *, 'Input frequency'
  read *, omega

  frequency_response = 1.0/cmplx(-omega*omega+ &
    1.0, 2.0*omega)
  real_part = real(frequency_response)
  imag_part = aimag(frequency_response)

! Calculate polar coordinates
! (magnitude and phase)

  magnitude = abs(frequency_response)
  phase = atan2(imag_part, real_part)
```

```
    print *, ' at frequency ', omega
    print *, 'response = ', real_part, ' + i ', &
      imag_part
    print *, 'in polar form'
    print *, ' magnitude = ', magnitude
    print *, ' phase = ', phase
  end program ch1502
```

15.4 Complex and Kind Type

The standard requires that there be a minimum of two kind types for real numbers and this is also true of the complex data type. Chapter 5 must be consulted for a full coverage of real kind types. We would therefore use something like the following to select a complex kind type other than the default:

```
integer , parameter :: &
   dp = selected_real_kind(15,307)
complex (dp) :: z
```

Chapter 21 includes a good example of how to use modules to define and use precision throughout a program and subprogram units.

15.5 Summary

Complex is used to store and manipulate complex numbers: those with a real and an imaginary part. There are standard functions which allow conversion between the numerical data types — `cmplx`, `real` and `int`.

15.6 Problem

15.1 The program used in Chap. 13 which calculated the roots of a quadratic had to abandon the calculation if the roots were complex. You should now be able to remedy this, remembering that it is necessary to declare any complex variables. Instead of raising the expression to the power 0.5 in order to take its square root, use the function `sqrt`. The formulae for the complex roots are

$$\frac{-b}{2a} \pm i\frac{\sqrt{-(b^2-4ac)}}{2a}$$

If you manage this to your satisfaction, try your skills on the roots of a cubic (see the problems in Chap. 13).