Chapter 42
Abstract Interfaces and Procedure Pointers

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

Albert Einstein

42.1 Introduction

We look at an example that illustrates the use of abstract interfaces and procedure pointers.

42.2 Example 1: Abstract Interfaces and Procedure Pointers

Abstract interfaces and procedure pointers were introduced into Fortran in the 2003 standard. One of the things their addition did was simplify the way we could program where several procedures shared a common interface.

Their addition also made it possible to solve problems involving procedure pointer components and abstract type bound procedures.

Here is some background technical material taken from the standard.

A procedure pointer is a procedure that has the EXTERNAL and POINTER attributes; it may be pointer associated with an external procedure, an internal procedure, an intrinsic procedure, a module procedure, or a dummy procedure that is not a procedure pointer.

An interface body in a generic or specific interface block specifies the EXTERNAL attribute and an explicit specific interface for an external procedure, dummy procedure, or procedure pointer.

A procedure declaration statement declares procedure pointers, dummy procedures, and external procedures. It specifies the EXTERNAL attribute for all entities in the proc-decl-list.

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_42

Here is the syntax for a procedure declaration statement.

```
PROCEDURE ( [ proc-interface ] )
  [ [ , proc-attr-spec ] ... :: ] proc-decl-list
```

We use abstract interfaces and procedure pointers in the example below.

```
module abstract_function_interface_module

  abstract interface
    real function f(i)
      implicit none
      integer , intent(in) :: i
    end function f
  end interface

end module abstract_function_interface_module

module fun01

  implicit none

contains

  real function f1(i)

    implicit none
    integer, intent (in) :: i

    f1 = 1.0/i

  end function f1

  real function f2(i)

    implicit none
    integer, intent (in) :: i

    f2 = 1.0/(i*i)

  end function f2

end module fun01

module fun02
```

```
  use abstract_function_interface_module

  contains

    real function f3(fun, i)

      implicit none
      integer, intent (in) :: i

      procedure (f) :: fun

      f3 = fun(i)

    end function f3

    real function f4(fun, i)

      implicit none
      integer, intent (in) :: i

      procedure (f) :: fun

      integer :: n
      real :: t

      t = 0.0

      do n = 1, 5
        t = t + fun(i)
      end do

      f4 = t

    end function f4

  end module fun02

  program ch4201

    use abstract_function_interface_module

    use fun01

    use fun02
    implicit none

    procedure (f) , pointer :: p1

    p1 => f1

    print *, ' p1 => f1, calling f3'
    print *, f3(p1, 2)

    p1 => f2

    print *, ' p1 => f2, calling f3'
    print *, f3(p1, 2)

    p1 => f1

    print *, ' p1 => f1, calling f4'
    print *, f4(p1, 2)

    p1 => f2

    print *, ' p1 => f2, calling f4'
    print *, f4(p1, 2)

  end program ch4201
```

Here is sample output.

```
  p1 => f1, calling f3
   0.5000000
  p1 => f2, calling f3
   0.2500000
  p1 => f1, calling f4
   2.5000000
  p1 => f2, calling f4
   1.2500000
```

42.3 Problem

42.1 Try out the example in this chapter.