Fortran is one of the oldest computer languages. It was developed in the 1950s by IBM for numeric calculations (Fortran is an abbreviation of "Formula Translation"). Despite its age, it is still used for high-performance computing such as weather prediction. However, the language has changed considerably over the years, although mostly maintaining backwards compatibility; well known versions are FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003, Fortran 2008, Fortran 2015, and Fortran 2018.

This overview will discuss the features of Fortran 95 since it is the most widely implemented of the more recent specifications and the later versions are largely similar (by comparison FORTRAN 77 is a very different language).

```
! This is a comment.


program example   !declare a program called example.

    ! Code can only exist inside programs, functions, subroutines or modules.
    ! Using indentation is not required but it is recommended.


    ! Declaring Variables
    ! ===================

    ! All declarations must come before statements and expressions.

    implicit none    !prevents dynamic declaration of variables (recommended!)
    ! Implicit none must be redeclared in every function/program/module...

    ! IMPORTANT - Fortran is case insensitive.
    real z
    REAL Z2

    real :: v,x    ! WARNING: default initial values are compiler dependent!
    real :: a = 3, b=2E12, c = 0.01
    integer :: i, j, k=1, m
    real, parameter :: PI = 3.1415926535897931    !declare a constant.
    logical :: y = .TRUE. , n = .FALSE.    !boolean type.
    complex :: w = (0,1)    !sqrt(-1)
    character (len=3) :: month    !string of 3 characters.

    real :: array(6)     !declare an array of 6 reals.
    real, dimension(4) :: arrayb    !another way to declare an array.
    integer :: arrayc(-10:10)   !an array with a custom index.
    real :: array2d(3,2)    !multidimensional array.

    ! The '::' separators are not always necessary but are recommended.

    ! many other variable attributes also exist:
    real, pointer :: p    !declare a pointer.
```

```fortran
integer, parameter :: LP = selected_real_kind(20)
real (kind = LP) :: d    !long precision variable.

! WARNING: initialising variables during declaration causes problems
! in functions since this automatically implies the 'save' attribute
! whereby values are saved between function calls. In general, separate
! declaration and initialisation code except for constants!


! Strings
! =======

character :: a_char = 'i'
character (len = 6) :: a_str = "qwerty"
character (len = 30) :: str_b
character (len = *), parameter :: a_long_str = "This is a long string."
!can have automatic counting of length using (len=*) but only for constants.

str_b = a_str // " keyboard"    !concatenate strings using // operator.


! Assignment & Arithmetic
! =======================

Z = 1    !assign to variable z declared above (case insensitive).
j = 10 + 2 - 3
a = 11.54  /  (2.3 * 3.1)
b = 2**3    !exponentiation


! Control Flow Statements & Operators
! ===================================

! Single-line if statement
if (z == a) b = 4  !condition always need surrounding parentheses.

if (z /= a) then !z not equal to a
! Other symbolic comparisons are < > <= >= == /=
  b = 4
else if (z .GT. a) then !z greater than a
! Text equivalents to symbol operators are .LT. .GT. .LE. .GE. .EQ. .NE.
  b = 6
else if (z < a) then !'then' must be on this line.
  b = 5 !execution block must be on a new line.
else
  b = 10
end if !end statement needs the 'if' (or can use 'endif').


if (.NOT. (x < c .AND. v >= a .OR. z == z)) then   !boolean operators.
  inner: if (.TRUE.) then    !can name if-construct.
```

```
      b = 1
    endif inner    !then must name endif statement.
  endif


  i = 20
  select case (i)
    case (0)    !case i == 0
      j=0
    case (1:10)    !cases i is 1 to 10 inclusive.
      j=1
    case (11:)    !all cases where i>=11
      j=2
    case default
      j=3
  end select


  month = 'jan'
  ! Condition can be integer, logical or character type.
  ! Select constructions can also be named.
  monthly: select case (month)
    case ("jan")
       j = 0
    case default
       j = -1
  end select monthly


  do i=2,10,2    !loops from 2 to 10 (inclusive) in increments of 2.
    innerloop: do j=1,3    !loops can be named too.
      exit    !quits the loop.
    end do innerloop
  cycle    !jump to next loop iteration.
  enddo


  ! Goto statement exists but it is heavily discouraged though.
  goto 10
  stop 1    !stops code immediately (returning specified condition code).
10  j = 201    !this line is labeled as line 10


  ! Arrays
  ! ======
  array = (/1,2,3,4,5,6/)
  array = [1,2,3,4,5,6]    !using Fortran 2003 notation.
  arrayb = [10.2,3e3,0.41,4e-5]
  array2d =  reshape([1.0,2.0,3.0,4.0,5.0,6.0], [3,2])

  ! Fortran array indexing starts from 1.
  ! (by default but can be defined differently for specific arrays).
  v = array(1)    !take first element of array.
  v = array2d(2,2)

  print *, array(3:5)    !print all elements from 3rd to 5th (inclusive).
  print *, array2d(1,:)    !print first column of 2d array.

  array = array*3 + 2    !can apply mathematical expressions to arrays.
  array = array*array    !array operations occur element-wise.
  !array = array*array2d    !these arrays would not be compatible.

  ! There are many built-in functions that operate on arrays.
  c = dot_product(array,array)    !this is the dot product.
  ! Use matmul() for matrix maths.
  c = sum(array)
  c = maxval(array)
  print *, minloc(array)
  c = size(array)
  print *, shape(array)
  m = count(array > 0)

  ! Loop over an array (could have used Product() function normally).
  v = 1
  do i = 1, size(array)
      v = v*array(i)
  end do

  ! Conditionally execute element-wise assignments.
  array = [1,2,3,4,5,6]
  where (array > 3)
      array = array + 1
  elsewhere (array == 2)
      array = 1
  elsewhere
      array = 0
  end where

  ! Implied-DO loops are a compact way to create arrays.
  array = [ (i, i = 1,6) ]    !creates an array of [1,2,3,4,5,6]
  array = [ (i, i = 1,12,2) ]    !creates an array of [1,3,5,7,9,11]
  array = [ (i**2, i = 1,6) ]    !creates an array of  [1,4,9,16,25,36]
  array = [ (4,5, i = 1,3) ]    !creates an array of [4,5,4,5,4,5]


  ! Input/Output
  ! ============

  print *, b    !print the variable 'b' to the command line

  ! We can format our printed output.
  print "(I6)", 320    !prints '   320'
  print "(I6.4)", 3    !prints '  0003'
  print "(F6.3)", 4.32    !prints ' 4.320'

  ! The letter indicates the expected type and the number afterwards gives
  ! the number of characters to use for printing the value.
  ! Letters can be I (integer), F (real), E (engineering format),
  ! L (logical), A (characters) ...
  print "(I3)", 3200    !print '***' since the number doesn't fit.

  ! we can have multiple format specifications.
  print "(I5,F6.2,E6.2)", 120, 43.41, 43.41
  print "(3I5)", 10, 20, 30    !3 repeats of integers (field width = 5).
  print "(2(I5,F6.2))", 120, 43.42, 340, 65.3   !repeated grouping of formats.

  ! We can also read input from the terminal.
  read *, v
  read "(2F6.2)", v, x    !read two numbers

  ! To read a file.
  open(unit=11, file="records.txt", status="old")
  ! The file is referred to by a 'unit number', an integer that you pick in
  ! the range 9:99. Status can be one of {'old','replace','new'}.
  read(unit=11, fmt="(3F10.2)") a, b, c
  close(11)

  ! To write a file.
  open(unit=12, file="records.txt", status="replace")
  write(12, "(F10.2,F10.2,F10.2)") c, b, a
  close(12)

  ! There are more features available than discussed here and alternative
  ! variants due to backwards compatibility with older Fortran versions.


  ! Built-in Functions
  ! ==================

  ! Fortran has around 200 functions/subroutines intrinsic to the language.
  ! Examples -
  call cpu_time(v)    !sets 'v' to a time in seconds.
  k = ior(i,j)    !bitwise OR of 2 integers.
  v = log10(x)    !log base 10.
  i = floor(b)    !returns the closest integer less than or equal to x.
  v = aimag(w)    !imaginary part of a complex number.


  ! Functions & Subroutines
  ! =======================

  ! A subroutine runs some code on some input values and can cause
  ! side-effects or modify the input values.
```

```
    call routine(a,c,v)    !subroutine call.

    ! A function takes a list of input parameters and returns a single value.
    ! However the input parameters may still be modified and side effects
    ! executed.

    m = func(3,2,k)  !function call.

    ! Function calls can also be evoked within expressions.
    Print *, func2(3,2,k)

    ! A pure function is a function that doesn't modify its input parameters
    ! or cause any side-effects.
    m = func3(3,2,k)


contains ! Zone for defining sub-programs internal to the program.

    ! Fortran has a couple of slightly different ways to define functions.

    integer function func(a,b,c)    !a function returning an integer value.
        implicit none   !best to use implicit none in function definitions too.
        integer :: a,b,c !type of input parameters defined inside the function.
        if (a >= 2) then
            func = a + b + c !the return variable defaults to the function name.
            return !can return the current value from the function at any time.
        endif
        func = a + c
        ! Don't need a return statement at the end of a function.
    end function func


    function func2(a,b,c) result(f)    !return variable declared to be 'f'.
        implicit none
        integer, intent(in) :: a,b    !can declare and enforce that variables
                                      !are not modified by the function.
        integer, intent(inout) :: c
        integer :: f     !function return type declared inside the function.
        integer :: cnt = 0    !GOTCHA - initialisation implies variable is
                              !saved between function calls.
        f = a + b - c
        c = 4    !altering the value of an input variable.
        cnt  = cnt + 1    !count number of function calls.
    end function func2


    pure function func3(a,b,c)  !a pure function can have no side-effects.
        implicit none
        integer, intent(in) :: a,b,c
        integer :: func3
        func3 = a*b*c
```

```fortran
    end function func3


    subroutine routine(d,e,f)
        implicit none
        real, intent(inout) :: f
        real, intent(in) :: d,e
        f = 2*d + 3*e + f
    end subroutine routine


end program example   ! End of Program Definition -----------------------


! Functions and Subroutines declared externally to the program listing need
! to be declared to the program using an Interface declaration (even if they
! are in the same source file!) (see below). It is easier to define them within
! the 'contains' section of a module or program.

elemental real function func4(a) result(res)
! An elemental function is a Pure function that takes a scalar input variable
! but can also be used on an array where it will be separately applied to all
! of the elements of an array and return a new array.
    real, intent(in) :: a
    res = a**2 + 1.0
end function func4


! Modules
! =======

! A module is a useful way to collect related declarations, functions and
! subroutines together for reusability.

module fruit
    real :: apple
    real :: pear
    real :: orange
end module fruit


module fruity
    ! Declarations must be in the order: modules, interfaces, variables.
    ! (can declare modules and interfaces in programs too).

    use fruit, only: apple, pear    ! use apple and pear from fruit module.
    implicit none    !comes after module imports.

    private    !make things private to the module (default is public).
    ! Declare some variables/functions explicitly public.
    public :: apple,mycar,create_mycar
    ! Declare some variables/functions private to the module (redundant here).
    private :: func4

    ! Interfaces
    ! ==========
    ! Explicitly declare an external function/procedure within the module
    ! (better in general to put functions/procedures in the 'contains' section).
    interface
        elemental real function func4(a) result(res)
            real, intent(in) :: a
        end function func4
    end interface

    ! Overloaded functions can be defined using named interfaces.
    interface myabs
        ! Can use 'module procedure' keyword to include functions already
        ! defined within the module.
        module procedure real_abs, complex_abs
    end interface

    ! Derived Data Types
    ! ==================
    ! Can create custom structured data collections.
    type car
        character (len=100) :: model
        real :: weight    !(kg)
        real :: dimensions(3)    !i.e. length-width-height (metres).
        character :: colour
    end type car

    type(car) :: mycar    !declare a variable of your custom type.
    ! See create_mycar() routine for usage.

    ! Note: There are no executable statements in modules.

contains

    subroutine create_mycar(mycar)
        ! Demonstrates usage of a derived data type.
        implicit none
        type(car),intent(out) :: mycar

        ! Access type elements using '%' operator.
        mycar%model = "Ford Prefect"
        mycar%colour = 'r'
        mycar%weight = 1400
        mycar%dimensions(1) = 5.0    !default indexing starts from 1!
        mycar%dimensions(2) = 3.0
        mycar%dimensions(3) = 1.5

    end subroutine
    real function real_abs(x)
        real :: x
        if (x<0) then
            real_abs = -x
        else
            real_abs = x
        end if
    end function real_abs

    real function complex_abs(z)
        complex :: z
        ! long lines can be continued using the continuation character '&'
        complex_abs = sqrt(real(z)**2 + &
                                         aimag(z)**2)
    end function complex_abs


end module fruity
```

More Resources

For more information on Fortran:

- wikipedia
- Fortran-lang Organization
- Fortran_95_language_features
- fortranwiki.org
- www.fortran90.org/
- list of Fortran 95 tutorials
- Fortran wikibook
- Fortran resources
- Mistakes in Fortran 90 Programs That Might Surprise You