Statements and expressions in Fortran

Victor Eijkhout, Susan Lindsey

Fall 2022

last formatted: August 28, 2022

Basics

1. Program structure

Structure of your program file:

```
Program foo
  < declarations >
  < statements >
End Program foo
```

2. Things to note

- No includes before the program.
- Program has a name (emacs tip: type `end<TAB>`)
- There is an `End`, rather than curly braces.
- Declarations first, not interspersed.

3. Comments

- Ignore to end of line

```
x = 1 ! set x to one
```

- comment after continuation

```
x = f(a) & ! term1
  + g(b)   ! term2
```

- No multi-line comments.

Compilation

Filename has extension `.F90` or `.f90`.
(this is modern 'free format'; the old 'fixed format' uses extensions `.F` or `.f`)

Compiler: `ifort` (Intel)
or `gfortran` (Gnu compiler collection)

4. Variable declarations

- Variable declarations at the top of the program unit, before any executable statements.
- declaration

  `type, attributes :: name1, name2, ....`

  where
  - *type* is most commonly `integer`, `real(4)`, `real(8)`, `logical`
  - *attributes* can be `dimension`, `allocatable`, `intent`, `parameter` et cetera.
- Keywords and variables are *case-insensitive*

5. Data types

- Numeric: `Integer`, `Real`, `Complex` (section **??**).
- Logical: `Logical`.
- Character: `Character`. Strings are realized as arrays of characters.
- Derived types: `Type`; chapter **??**.

6. Variable kind and range

If you want to know what the kind and range of a type is, use `kind` and `huge`:

Code:

```
Integer :: idef
Real :: rdef
Real(8) :: rdouble

print 10,"integer is kind",kind(idef)
print 10,"integer max is",huge(idef)
print 10,"real is kind",kind(rdef)
print 15,"real max is",huge(rdef)
print 10,"real8 is kind",kind(rdouble)
print 15,"real8 max is",huge(rdouble)
```

Output

```
[typef] def:
 integer is kind
           4
  integer max is
    2147483647
    real is kind
           4
     real max is
    0.3403E+39
   real8 is kind
           8
    real8 max is
    0.1798+309
```

7. Complex

Complex constants are written as a pair of reals in parentheses. There are some basic operations.

Code:

```
Complex :: &
     fourtyfivedegrees = (1.,1.), &
     number,rotated
Real :: x,y
print *,"45
     degrees:",fourtyfivedegrees
x = 3. ; y = 1.; number = cmplx(x,y)
rotated = number * fourtyfivedegrees
print '("Rotated number has
     Re=",f5.2," Im=",f5.2)',&
     real(rotated),aimag(rotated)
```

Output

```
[basicf] complex:

45 degrees:

     (1.00000000,1.00000000)
Rotated number has
     Re= 2.00 Im= 4.00
```

8. Strings

```
character*20 :: prompt
prompt = "up to 20 characters"
```

9. Implicit typing

Fortran strictly does not need variable declarations (like python): variables have a type that is determined by name.
This is legal:

```
Program danger
  i = 3
  x = 2.5
  j = i * x
  print *,y
end Program danger
```

But:
This is **very dangerous**. Use `implicit none` in every program unit.

```
Program myprogram
  implicit none
  integer :: i
  real :: x
  ! more stuff
End Program myprogram
```

10. Boolean expressions

- Long form:
  `.and.  .not.  .or.`
  `.lt.  .le.  .eq.  .ne.  .ge.  .gt.`
  `.true.  .false.`
- Short form:
  `< <= == /= > >=`

11. Parameter

Variable=name+value.
What if the value should never change? Use the `parameter` attribute

```
real,parameter :: pi = 3.141592
```

This can not be changed like an ordinary variable.

Statements

12. Fortran statements

It's all much like C++:

- Assignments:

```
x = y
x = 2*y / (a+b)
```

  (Note the lack of semicolons at the end of statements.)
- I/O: next
- conditionals and loops: later
- function calls: later

13. Statements

- One line, one statement

```
x = 1
y = 2
```

  (historical reasons)

- semicolon to separate multiple statements per line

```
x = 1; y = 2
```

- Continuation of a line

```
x = very &
  long &
  expression
```

14. Simple I/O

- Input:

  ```
  READ *,n
  ```

- Output:

  ```
  PRINT *,n
  ```

There is also `Write`.

The 'star' indicates that default formatting is used.
Other syntax for read/write with files and formats.

15. Arithmetic expressions

- Pretty much as in C++
- Exception: `r**a` for power r^a.
- Modulus is a function: `MOD(7,3)`.

Exercise 1

Write a program that :

- displays the message `Type a number`,
- accepts an integer number from you (use `Read`),
- makes another variable that is three times that integer plus one,
- and then prints out the second variable.

Optional exercise 2

Write two programs, one that reads a temperature in Centigrade and converts to Fahrenheit, and one that does the opposite conversion.

$$C = (F - 32) \cdot 5/9, \qquad F = 9/5\, C + 32$$

Check your program for the freezing and boiling point of water. (Do you know the temperature where Celsius and Fahrenheit are the same?)

Can you use Unix pipes to make one accept the output of the other?