Statements and expressions in Fortran

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

Basics

Program structure

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

Statements

- One line, one statement

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

- semicolon to separate multiple statements per line

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

- Continuation of a line

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

Comments

- Ignore to end of line

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

- comment after continuation

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

Variable declarations

- Variable declarations at the top of the problem
- Variables are implicitly defined. Dangerous, so use:

  ```
  implicit none
  ```

- declaration

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

  where
  - *type* is most commonly `integer`, `real(4)`, `real(8)`, `logical`
  - *attributes* can be `dimension`, `allocatable`, `intent`, `parameters` et cetera.

Implicit typing

Fortran does not need variable declarations:
type are determined by name.
This is very dangerous. Use:

```
implicit none
```

in every program unit.

Single precision constants

```
real(8) :: x
x = 3.14
y = 6.022e-23
```

Double precision constants

- Use a compiler flag such as `-r8` to force all reals to be 8-byte.
- Write `3.14d0`
- `x = real(3.14, kind=8)`

Floating point types

Indicate number of bytes:

```
integer(2) :: i2
integer(4) :: i4
integer(8) :: i8

real(4) :: r4
real(8) :: r8
real(16) :: r16

complex(8) :: c8
complex(16) :: c16
complex*32 :: c32
```

Numerical precision

Number of bytes determines numerical precision:

- Computations in 4-byte have relative error $\approx 10^{-6}$
- Computations in 8-byte have relative error $\approx 10^{-15}$

Also different exponent range: max 10^{50} and 10^{300} respectively.

Complex

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

Code:

```
Complex :: fourtyfivedegrees = (1.,1.), &
     other
print *,fourtyfivedegrees
other = 2*fourtyfivedegrees
print *,other
```

Output from running complex in code directory basicf:

```
(1.00000000,1.00000000)
(2.00000000,2.00000000)
```

Arithmetic expressions

- Pretty much as in C++
- Exception: `r**2` for power.
- Modulus is a function: `MOD(7,3)`.

Boolean expressions

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

Statements

I/O routines

- Input:
  ```
  READ *,n
  ```
- Output:
  ```
  PRINT *,n
  ```

There is also `WRITE`.

Other syntax for read/write with files and formats.

Exercise 1

```
echo 100 | ./c2f | awk '/Equivalent/ {print $3}' | ./f2c
```

Optional exercise 2

```
echo 100 | ./c2f | awk '/Equivalent/ {print $3}' | ./f2c
```