

The Fortran 90 programming language

- Fortran has evolved since the early days of computing
- Fortran 90/95 is a modern programming language
- Many useful features for scientific (numerical) computing
- Widely used language in computational science
 - also widely used in finance, engineering, etc.
- Many “canned” subroutines available (often Fortran 77)
- Relatively easy to learn

Fortran 90 compilers

- Many commercial compilers available; often expensive
 - f90 available on buphy (Physics Dept. server)
- g95; free open source Fortran 90/95 compiler
 - available on CAS workstations
 - can be downloaded at www.g95.org
 - installed on Physics Dept server buphy (and buphy0)
- gfortran; other open source product

Fortran 90 language tutorial

- Introduction to basic elements needed to get started
- Simple examples used to illustrate concepts
- Example programs also available on the web site
- For more complete language description, see, e.g.,
 - [Fortran 90/95 explained](#), by M. Metcalf and J. Reid
 - [Fortran 90/95 for Scientists and Engineers](#), by S. Chapman

Discussion and practice at Friday tutorials

To create a Fortran 90 program:

- Write program text to a file, using, e.g., Emacs

```
[program program-name]
  program statements
  ...
end [program program-name]
```

- Compile & link using Fortran 90 compiler
 - creates “object” (.o) files
 - object files linked to form executable (.out or .x) file

Compilation/linking (using g95)

- > `g95 program.f90`
(gives executable program a.out)
- > `g95 program.f90 -o program.x`
(gives executable named program.x)
- > `g95 -O program.f90`
(turns on code optimization)
- > `g95 -O program1.f90 program2.f90`
(program written in more than one file)
- > `g95 -O program1.f90 program2.o`
(unit program2 previously compiled)

Variables and declarations

Intrinsic variable types

- real (floating-point)
- integer
- complex
- logical (boolean)
- character, character string (“text”)
- arrays of all of these (up to 7-dimensional)

- A declaration is used to state the type of a variable
- Without declaration, `real` assumed, except for variables with names starting with `i,...,n`, which are `integer`
- Declarations forced with `implicit none` statement
 - always use this; eliminates 99% of programming errors!
- Fortran 90 is case-insensitive

Floating-point numbers

A simple program which assigns values to real variables (single- and double precision; use 4 and 8 bytes):

```
implicit none

real    :: a
real(8) :: b

a=3.14159265358979328
print*,a
b=3.14159265358979328
print*,b
b=3.14159265358979328d0
print*,b

end
```

Output:

```
3.14159274
3.1415927410125732
3.1415926535897931
```

3.14159265358979328_8

is another way to specify double precision (8 bytes)

Integers

A standard integer uses 4 bytes, holds numbers -2^{31} to $2^{31}-1$

```
integer :: i
```

```
i=-2**31
```

```
print*,i
```

```
i=i-1
```

```
print*,i
```

Output:

```
-2147483648
```

```
2147483647
```

A “long” integer, `integer(8)`, is 8 bytes, holds -2^{63} to $2^{63}-1$

“Short” integers: `integer(2)`, `integer(1)`

Integer division: $3 / 2 = 1$, but $3 . / 2 = 1 . 5$

Complex numbers

Assignment of real and imaginary parts: $a = (a_r, a_i)$

```
complex :: a
```

```
a=(1.,2.)
```

```
print*,a,real(a),aimag(a)
```

```
a=a*(0.,1.)
```

```
print*,a
```

Output:

```
(1.,2.), 1., 2.
```

```
(-2.,1.)
```

`real(a)` and `aimag(a)` extract real and imaginary parts

Double precision: `complex(8)`

Logical (boolean) variables

Values denoted as `.true.` and `.false.` in programs
In input/output; values are given as T and F

Examples of boolean operators:

and, or, neqv (same as exclusive-or), not:

```
logical :: a,b
```

```
print*,'Give values (T/F) for a and b'
```

```
read*,a,b
```

```
print*,a.or.b,a.and.b,a.eqv.b,.not.a,.not.b
```

Running this program \Rightarrow

```
Give values (T/F) for a and b
```

```
T F
```

```
T F F F T
```