Preparation

- **Log in to mirage**

  ```
  $ ssh -l login mirage[0-2].ucar.edu
  ```

  Use CryptoCard or Yubikey

- **Copy the example source files**

  ```
  $ cp –r /glade/home/dnagle/f90-1 .
  ```

Modern Fortran I for Computational Scientists

Consulting Services Group
Si Liu & Dan Nagle (Presenter)
May 22, 2012

CISL Computational & Information Systems Laboratory
NCAR

Outline

- Brief Introduction
- Hello World
- Types, Assignments, Attributes
- Control Structures
- Input / Output
- Procedures Basics

Introduction

- Oldest Successful Language
- First International Standard Language
- Modern, Object-Oriented, Numerical, Parallel Programming
- Highly Portable

History of Fortran

- IBM Formula Translator (1957)
- Fortran 66 (1967) first standard
- Fortran 77 (1978)
- Fortran 90 (1991)
- Fortran 95 (1997)
- Fortran 2003 (2004)
- Fortran 2008 (2010)

01-Hello_World

- use the compiler to enforce the standard for best portability and bug catching
- compile using all warnings
- compile using full checking (at least during development)
- case insensitive (so don't use case!)
- program .. end program is a scoping unit

free form source

- lines up to 132 characters
- lowercase and uppercase are required to be treated identically
- names up to 63 characters
- comments may follow exclamation (!)
- ampersand (&) is a continuation symbol
- file name (usually!) has ".f90" suffix

Hello_World.mk

- set the compiler and compiler options
- specify the linker and linker options
- linker options will include libraries
- make rules controls when to compile
- commands control how to compile
- commands start with a **tab**

02-Hello_World

- intrinsic modules are standard
- intrinsic environment module specifies processor-dependencies
- named constants help avoid inconsistent values
- specific format gives control of the output text

03-Hello_World

- version control gives identification
- procedures encapsulate common tasks
  - makes program flow easier to follow
  - helps reuse
- compiler_version, compiler_options identify the compiler and options

version control

- use rcs if unsure what to use
- very simple, only three commands:
  - ci: create an archive from a source or place a new version in an existing archive
  - co: get the most recent version read-only
  - co –l: get the most recent version read-write for editing
- can also co a specific version

04-Integer

- inquiry intrinsic procedures give properties of any kind of integer
- integers have all theusual operations, including exponentials and two varieties of modulus
- integers use the i*w* format descriptors

05-Integer

- kind values parameterize types
- kind values are available from the intrinsic environment module
- one kind must support a decimal range of at least 10^{18}
- do not use literal kind values!

06-Character

- the default character is likely the only one supported
- assignment of fixed-length characters is truncate or blank-fill
- most compilers use 7-bit ascii
- many can read or write utf-8
- use a*w* format descriptors

07-Character

- the character operator is the concatenation operator //
- intrinsics: len(), trim(), len_trim()
- searches: index(), scan(), verify()
- conversion between character and other types is via internal reads and writes

08-Logical

- almost never use non-default logical
- has one unary operator: .not.
- has four binary operators:

  .and., .or., .eqv., .neqv.
- uses l*w* format descriptors

09-Logical

- relational operators: ==, /=, <. <=, >, >=
- operate between integer, real or character operands
- == and /= also operate between complex operands
- kinds or types are cast as needed

10-Real

- default real is usually 32-bit
- use inquiry intrinsics to learn about real kinds
- operators: + - * / **
- formats: f*w.d* e*w.d* es*w.d* en*w.d* format descriptors

11-Real

- must have two kinds of rea;
  - usually, real32 and real64
  - most compilers also have real128
- get kind values from environment module
- may also have x86 80-bit real
  - might not be in memory!

12-Complex

- must have a complex kind for every real kind
  - usually, real32 and real64
  - most compilers also have real128
- use same format descriptors as real
  - but must use two for each quantity

13-Derived_Type

- derived types are defined by the program
- derived types have a default assignment, but all other operators must be defined
- derived types have default value constructors

14-Assignment

- automatic type conversion among the numeric types in expressions and assignment
- tries to preserve values
- but may not be able to do so

15-Assignment

- assignment works for scalars or arrays
- arrays must have the same shape (and the operation must otherwise make sense)
- can use where
- can use forall

16-Attributes

- parameter makes a named constant
- can be any type and kind
- can be scalar or array
- save makes persistent values
- initial values implies save!

17-Attributes

- target means a variable may have aliases
- pointer makes a name an alias for other names
- a pointer to a pointer points to the target!

18-Attributes

- allocatables are allocated and deallocated
- if unsaved, al allocatable is automatically deallocated at procedure return
- so allocatables can be safe pointers

19-Attributes

- a pointer is a true alias
- so compilers treat them cautiously
- pointers may be allocated and deallocated
- but always use an allocatable variable wherever possible

20-Attributes

- an array is a sequence of values of the same type and kind
- arrays may be assigned to each other provided that are the same shape
- arrays may be operands provided they are the same shape

21-Attributes

- a coarray is the value of the same type and kind on a different image
- coarrays may be ordered as arrays are
- coarrays have are designed so compilers won't move them

22-Executables

- continue is a no-operation
- call and return invoke procedures and return from them
- if is the primary choice executable statement
- do is the primary iteration executable statement

23-Executables

- select case is used when the choices are represented by constants
- may also use sets of constants
- or ranges of constants

24-Executables

- do loops come in three varieties
- do forever needs an exit statement
- do concurrent is not yet widely implemented
- counted do often interacts with array bounds

25-Input_Output

- integers called logical units represent connections to files
- input_unit is the standard input
- output_unit is the standrad output
- error_unit is the standard error
- other files must be opened

26-Input_Output

- open associated a unit with a file
- the interaction of the open with the file system is controlled by the status specifier
- old means the file must exist
- new means the file must not exist
- replace means overwrite

27-Input_Output

- the access method controls how data is moved between the file and internal storage
- sequential; access and direct access are record oriented
- stream access is file storage unit oriented

28-Input_Output

- direct access must have a record length
- a file may be formatted or unformatted
- a status may be scratch
- scratch files are nameless and are deleted when closed

29-Input_Output

- namelist is a formatted input/ output scheme whereby variables are named in the file
- namelist is suitable for small input files such as initialization files
- namelist is almost an input langauge

30-Procedures

- procedures have interfaces
- which may be implicit or explicit
- most modern features require an explicit interface
- internal procedures and module procedures have explicit interfaces

31-Procedures

- a procedure may be recursive
- put the recursive keyword on the procedure statement
- mind the terminal condition to stop the recursion!

32-Procedures

- when a procedure is referenced, the actual arguments are associated with the dummy arguments
- an argument may be optiona;
- a procedure may be pure
  - no side effects

33-Procedures

- an array may be passed as an assumed shape array
- the called procedure is written to handle any shape array
- should also use the contiguous attribute where supported
- use inquiry intrisnics to get shape

Modern Fortran I for Computational Scientists

Thanks for Attending!

May 22, 2012