Overview

- Presented by Mark Branson, Don Dazlich and Ross Heikes
- Presentation materials and sample codes available at the web site:
  
  kiwi.atmos.colostate.edu/fortran
Intended Audience

• Some people will already know some Fortran
• Some people will be programmers in other languages
• Some people will be complete newcomers

This course is intended for all three groups!
Beginnings

• Fortran does not have a command-line interpreter like IDL and Matlab.

• You need an editor to write the code in and a Unix or Linux shell window to compile and execute it.

• Each individual will need to determine the compiler that’s available on their system.
Classes of Language

Interpreted

Shell script
Perl
Python

Compiled

Java
C, C++, Fortran

*Fortran is the best choice for pure number crunching!*
FORmula TRANslator invented 1954-8 by John Backus and his team and IBM

general purpose programming language mainly intended for mathematical computations in engineering

first-ever high-level programming language using the first compiler ever developed
History (2)

FORTRAN 66 (ISO Standard 1972)
FORTRAN 77 (1980) \{ HUGE TRANSITION! \}
Fortran 90 (1991)
Fortran 95 (1996)
Fortran 2008 (ongoing)
Disclaimer

This course will cover modern, free-format Fortran only!

• Don’t want to teach newcomers “old” fortran.

• At the same time almost all of you already have or will encounter your fair share of legacy Fortran codes.

• Almost all old Fortran remains legal.
Hardware and Software

A system is built from **hardware** and **software**

The **hardware** is the physical medium
- CPU, memory, keyboard, display

The **software** is a set of computer programs
- operating system, compilers, editors
- Fortran programs
Programs

Fortran 90 is a high-level language
Uses English-like words and math expressions

Y = X+3
PRINT *, Y

Compilers translate into machine instructions
A linker then creates an executable program
The operating system runs the executable
Algorithms and Models

An algorithm is a set of instructions. They are executed in a defined order. Doing that carries out a specific task.

The above is known as *procedural programming*. Fortran 90 is a *procedural language*.

*Object-orientation* is still procedural. Fortran 90 has *object-oriented* facilities.
An Example of a Problem

Write a program to convert a time in hours, minutes and seconds to one in seconds only.

Algorithm:
1. Multiply the hours by 60.
2. Add the minutes to the result.
3. Multiply the result by 60.
4. Add the seconds to the result.
Logical Structure

1. Start of program
2. Reserve memory for data
3. Write prompt to display
4. Read the time in hours, minutes and seconds
5. Convert the time into seconds
6. Write out the number of seconds
7. End of program
The Program

PROGRAM example1
! comments start with an exclamation mark
IMPLICIT NONE
INTEGER :: hours, mins, secs, temp
PRINT *, 'Type in the hours, minutes and seconds'
READ *, hours, mins, secs
temp = 60*(hours*60 + mins) + secs
PRINT *, 'Time in seconds = ',temp
END PROGRAM example1
High Level Structure

1. Start of program (or procedure)
   PROGRAM example1

2. Specification part
   Declare types and sizes of data

3. - 6. Execution part
   All of the “action” statements

7. End of program (or procedure)
   END PROGRAM example1
Program and File Names

- The program and file names are NOT related. PROGRAM QES can be in the file QuadSolver.f90

Some implementations like the same names, sometimes converted to lower- or upper-case.

The compiler documentation should tell you!
The Specification Part

Reserve memory for data

    INTEGER :: hours, mins, secs, temp

INTEGER is the type of the variables

hours, mins, secs are used to hold input

The values read in are called the input data

temp is called a workspace variable (also called a temporary variable)

The output data are ‘Time... =’ and temp

They can be any expression not just a variable
Write prompt to display

PRINT *, ‘Type the hours, …’

Read the time in hours, minutes and seconds

READ *, hours, mins, secs

Convert the time into seconds

\[
\text{temp} = 60 \times (\text{hours} \times 60 + \text{mins}) + \text{sec}
\]

Write out the number of seconds

PRINT *, ‘Time in seconds = ‘, temp
Compiling and Executing

Compile your program into an executable:

```
f90 [-o exename] program_name.f90
```

where

```
f90 = name of your compiler (f90, ifort, gfortran, g90, etc.)
```

If you do not specify an executable most systems will use `a.out` by default.
Really Basic I/O

READ *, <variable list> reads from stdin
PRINT *, <expression list> writes to stdout

Both do input/output as human-readable text
Each I/O statement reads/writes on a new line

A list is items separated by commas
Variables are anything that can store values
Expressions are anything that can deliver a value
Developing a Computer Program

There are four main steps:

1. Specify the problem
2. Analyze and subdivide into tasks
3. Write the Fortran 90 code
4. Compile and run (testing phase)

Each step may require several iterations.
You may need to restart from an earlier step.
The testing phase is very important.
Errors

• If the syntax is incorrect, the compiler says so

    INTEGER :: ,mins,secs

• If the action is invalid, things are messier

    \( X / Y \) when \( Y \) is zero

    Error message at run-time OR
    Program may crash or hang or produce nonsense values
Fortran Language Rules

• This course is modern, free-format source only
• Almost all old Fortran remains legal BUT you should avoid using it as modern Fortran is better
Important Warning

- Fortran **syntax** (the arrangement of words and phrases) is verbose and horrible. It can fairly be described as a historical mess.

- Fortran **semantics** (the mean of words, phrases, or text) are fairly clean and consistent.

- Verbosity causes problems for examples. Many use poor style to be readable, lack error checking.

- **DO WHAT I SAY NOT WHAT I DO**
Correctness

Humans understand language quite well even when it isn’t strictly correct.

Computers (i.e., compilers) are not so forgiving.

- Programs must follow the rules to the letter.

Fortran compilers will flag all syntax errors. Good compilers will detect more than is required.

But your error may just change the meaning OR do something invalid (“undefined behavior”).
Examples of Errors

Consider \((N \times M/1024+5)\)

If you mistype the ‘0’ as a ‘)’: \((N \times M/1)24+5)\)

You will get an error message when compiling. It may be confusing but will point out a problem.

If you mistype the ‘0’ as a ‘-’: \((N \times M/1-24+5)\)

You will simply evaluate a different formula and get wrong answers with no error message.

And if you mistype ‘*’ as ‘8’?
Character Set

Letters (A to Z and a to z) and digits (0 to 9)
Letters are matched ignoring their case

And the following special characters
_ = + - * / ( ) , . ’ : ! ” % & ; < > ? $
Plus space (i.e., a blank) but not tab
The end-of-line indicator is not a character

Any character allowed in comments and strings
• Case is significant in strings and only there
Special Characters

_ = + - * / ( ) , . ’ : ! ” % & ; < > ? $ 

slash (/) is also used for divide 
hyphen (-) is also used for minus 
asterisk (*) is also used for multiply 
apostrophe (‘) is also used for single quote 
period (.) is also used for decimal point 

The others are described when we use them.
Source Form (1)

Spaces are not allowed in keywords or names
INTEGER is not the same as INT EGER

HOURS is the same as hours or hoURs
But not HO URS - that means HO and URS

Some keywords can have two forms:
ENDDDO is the same as END DO
But EN DDO is treated as EN and DDO
• Do not run keywords and names together
  PROGRAMMyPROG - illegal
  PROGRAM  MyPROG - allowed

• You can use spaces liberally for clarity
  INTEGER :: I, J, K
  Exactly where you use them is a matter of taste

• Blank lines can be used in the same way as well
  as well as lines consisting only of comments
Lines and Comments

A line is a sequence of up to 132 characters.

A comment is from ! to the end of the line. The whole of a comment is totally ignored by the compiler.

\[ A = A + 1 \quad ! \text{These characters are ignored} \]

\[ ! \text{That applies to },", \& \text{ and } ; \text{ too.} \]

Blank lines are completely ignored.

\[ ! \quad ! \text{Including ones that are just comments} \quad ! \]
Use of Layout

• Well laid-out programs are must more readable.
• You are less likely to make trivial mistakes AND much more likely to spot them.
• This also applies to low-level formats, too.

1.0e6 is clearer than 1.e6 or .1e7
Use of Comments

- Appropriate commenting is very important.
- Document assumptions that may break later.
- Also helps to remind you to not make the same mistake twice!
- Good commenting can slow coding by 25% BUT it really speeds up initial debugging!
- Overall in research it repays itself 3:1. Can be 10:1 for production codes.
Use of Case

• It doesn’t matter which case convention you use **BUT** do try to be moderately consistent.

• Very important for clarity and editing/searching.

• One possible convention:
  • **UPPER** case for **keywords**
  • **Lower** case for **names**
Statements and Continuation

• A **program** is a sequence of **statements** used to build high-level constructs.

• **Statements** are made up out of **lines**.

• **Statements** are continued by appending `&`

\[
A = B + C + D + E + \& F + G + H
\]

is equivalent to

\[
A = B + C + D + E + F + G + H
\]
Other Rules (1)

• Statements can start at any position.

• Use indentation to clarify your code.

```plaintext
IF (a > 1.0) THEN
  b = 3.0
ELSE
  b = 2.0
END IF
```

• A number starting a statement is a label.

```plaintext
10 A = B + C
```

The use of labels is described later.
Other Rules (2)

**Semi-colons** can be used to put **multiple statements** on the same line:

```plaintext
a = 3 ; b = 4 ; c = 5
```

Overusing that can make a program unreadable BUT it can clarify your code in some cases.

Avoid mixing continuation with that and comments. It is legal but makes code VERY hard to read.

```plaintext
a = b + c ; d = e + f + &
g + h
a = b + c + & ! More coming...
```
Breaking Character Strings

Continuation lines can start with an \& Preceding spaces and the \& are suppressed.

The following works and allows indentation:

PRINT *, ‘Assume that this string \& \&is far too long and complic\& \&ated to fit on a single line’

The initial \& avoids including excess spaces AND avoids problems if the text starts with !

This may also be used to continue any line.
Names

- Up to **31** letters, digits and underscores.
- Names must start with a letter.
- Upper and lower case are equivalent.
  
  DEPTH, Depth and depth are all the same.
- The following are valid fortran names:

  A, AA, aaa, Tax, INCOME, Num1, Num2, NUM333, N12MO5, atmospheric_pressure, Line_Color, R2D2, A_21_173_5a
Invalid Names

The following are invalid names:

1A does not begin with a letter
_B does not begin with a letter
Depth$0 contains an illegal character ‘$’
A-3 would be interpreted as subtract 3 from A
B.5: contains illegal characters ‘.’ and ‘:’