Fortran Seminar Series

Fall 2022

Overview

- Presented by Mark Branson and Don Dazlich
- Presentation materials and sample codes available at the web site:

  hogback.atmos.colostate.edu/fortran

  - ***Kelley keeps this updated on a weekly basis***

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!

Why Fortran?

- Almost ***every major model*** in atmospheric and oceanic science is still written in Fortran (CAM or CESM, RAMS, WRF, ECMWF's suite of models, NWP, etc.)
- Fortran has a reputation for being hopelessly out of date (mainly due to Fortran 77?)
- No courses offered except in Meteorology departments

Speed Test

Solve 2D Laplace equation with Jacobi interactive solver

$$U_{xx} + U_{yy} = 0$$

using a fourth-order compact finite difference scheme

$$U_{ij} = (4(U_{i-1,j} + U_{i,j-1} + U_{i+1,j} + U_{i,j+1}) + U_{i-1,j-1} + U_{i+1,j-1} + U_{i+1,j+1} + U_{i-1,j+1}) / 20$$

Dang It's Fast!

Results with different software (Execution time in seconds)

Compiler/Package	n=50	n=100
Python	46.15	751.78
NumPy	0.61	6.39
Matlab	0.64	6.53
Java	0.12	2.2
gfortran	0.24	3.25
ifort	0.052	0.66

Courses

- CSU Atmos used to have a programming course (Fortran/UNIX, IDL and Matlab)

  http://www.atmos.colostate.edu/programming/ 

- Iowa State has a Fortran/Python course

  http://www.meteor.iastate.edu/classes/mt227/

- Univ of Miami Scientific Programming course 

  http://www.rsmas.miami.edu/personal/miskandarani/Courses/MSC321/

Hands On!

- In the past we've just lectured and not done any "hands on" work.
- But the best way to learn any programming language is to get some hands on experience.
- We'd like to try splitting each session into lecture and hands-on.
- Use your own compiler on your laptop

GREAT RESOURCE: fortran-lang.org

Proposed Syllabus

1. Beginnings
2. Data Types and Basic Calculation
3. Control Constructs
4. Array Concepts
5. Subroutines and Functions
6. Modules
7. Parameterized Data Types

Proposed Syllabus (cont.)

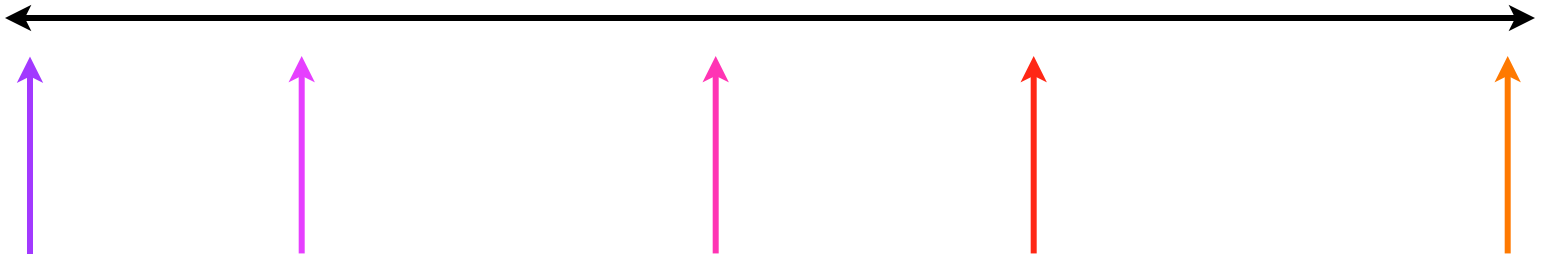
8. Input and Output (Don)
9. Derived Types
10. Computer Arithmetic (Don)
11. Make and Makefiles
12. Introduction to Parallel Programming (Don)

Beginnings

- Fortran does not (yet) have a command-line interpreter like IDL, Matlab or Python.
- 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 **Compiled**



Shell script Perl Python Java C,C++,Fortran

Fortran is the best choice for pure number crunching!

History

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)

Fortran 90 (1991)

} ***HUGE TRANSITION!***

Fortran 95 (1996)

Fortran 2003 (2004)

Fortran 2008 (2010)

Fortran 2018 (formerly Fortran 2015)

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 temperature value in degrees Fahrenheit to degrees Celsius and degrees Kelvin.

Algorithm:

1. Read in the initial temperature value (F).
2. Apply equation to compute temp in C.
3. Apply equation to compute temp in K.
4. Write out the results.

Logical Structure

1. Start of program
2. Reserve memory for data
3. Write prompt to display
4. Read in the initial temperature value (F)
5. Convert to both C and K
6. Write out the results
7. End of program

The Program

temp_conversion.f90 (break out of keynote)

High Level Structure

1. Start of program (or procedure)

   PROGRAM temp_conversion

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 temp_conversion

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

```
REAL :: deg_f, deg_c, deg_k
```

REAL is the type of the variables

The variable deg_f is used to hold input

The value read in is called the input data

The output data are the string, deg_c and deg_k

They can be any expression not just a variable

The Execution Part

Write prompt to display

```
PRINT *, 'Please type in the temp in F'
```

Read the temperature in degrees F

```
READ *, deg_f
```

Convert to Celsius and Kelvin

```
deg_c = 5.*(deg_f-32.)/9.

deg_k = deg_c + 273.15
```

Write out the results

```
print*, "This is equal to", deg_c, "C and",deg_k, "K"
```

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

Example

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

- ALWAYS keep in mind the ***golden rule***:

  Computers ONLY do what you tell them to do.

  - If something is wrong, it's probably your own fault. I'm sorry, but it is.
  - Corollary: Sometimes you don't know that you told the computer to do it wrong, OR somebody else did the telling.

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 understad linguage quite well even when it isnt stroctly correc

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*M/1024+5)

If you mistype the '0' as a ')': (N*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*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:

ENDDO is the same as END DO

But EN DDO is treated as EN and DDO

Source Form (2)

- 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   ! These characters are ignored
! That applies to !, & and ; too.
```

Blank lines are completely ignored.

```
!
!  Including ones that are just comments
!
```

Use of Layout

- Well laid-out programs are much 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.

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

- A number starting a statement is a label.

```
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:

```
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.

```
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 ':'