Fortran 90 Seminar

Spring 2009

Overview

- Presented by Mark Branson, Ross Heikes and Don Dazlich
- Everything from the basics to advanced topics like makefiles, optimization, parallelization.
- Suggestions from the audience???
- Presentation materials and example codes will be made available at the website. (kiwi/fortran)

Let's Jump Right In!

* Need an editor to write the program code in and a Unix shell window to compile it.
* Fortran does not have a command-line interpreter like IDL and Matlab.
* How do I know the name of my compiler and where it's located?

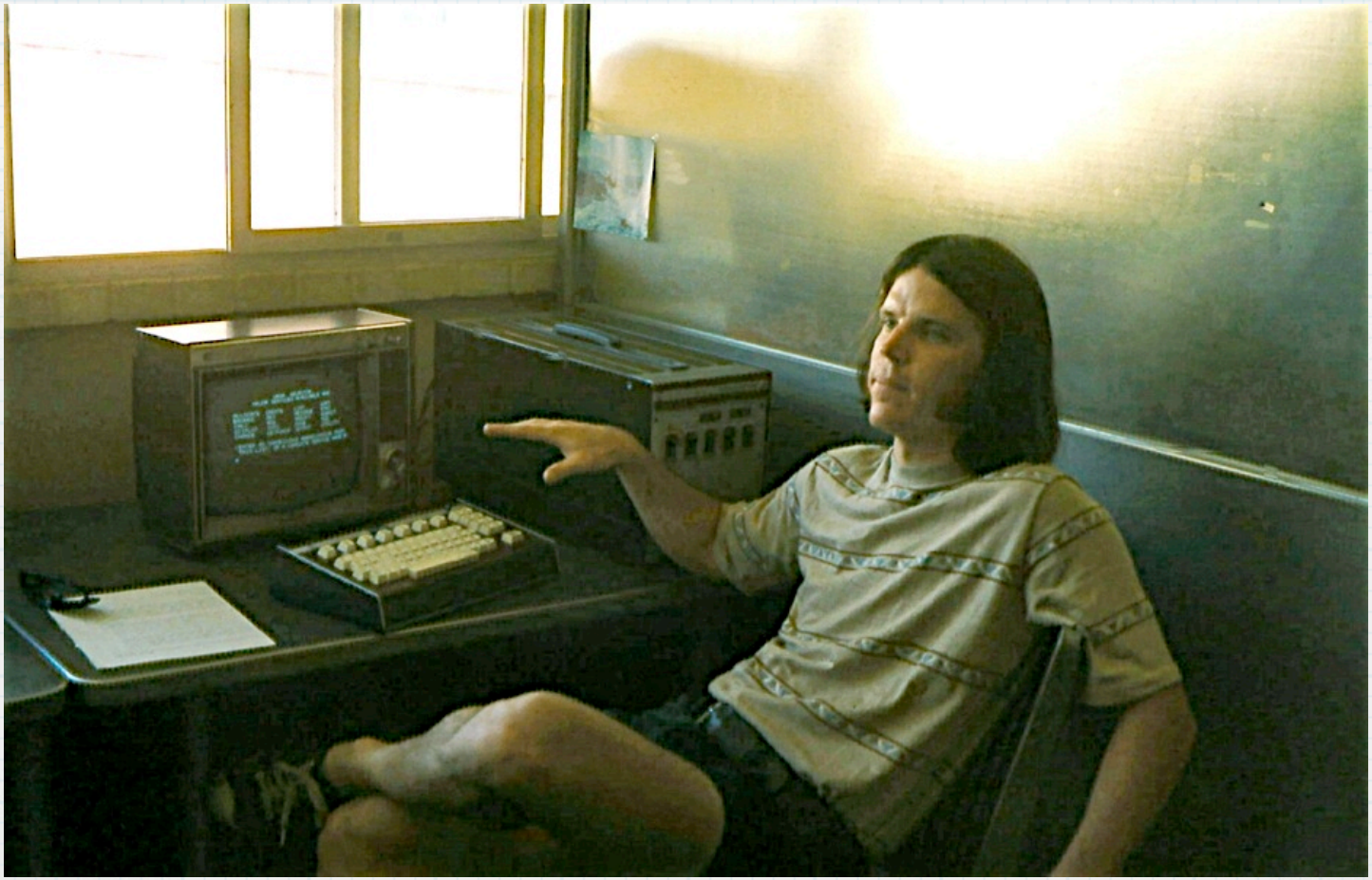
What is Fortran?

* general-purpose programming language mainly intended for mathematical computations in engineering.
* Fortran is an acronym for **FOR**mula **TRAN**slator
* first-ever high-level programming language, using the first compiler ever developed
* initially developed by a team of programmers at IBM lead by John Backus, and first published in **1957**.

Why learn Fortran?

* Fortran is the dominant programming language used in engineering applications AND the most enduring computer programming language in history!
* From time to time, experts have predicted the extinction of Fortran, and these predictions have always failed.
* One of the main reasons it has endured: software inertia.
  * Reliable software translation is very difficult and EXPENSIVE!

History of Fortran



History of Fortran (2)

* Fortran I : contained 32 statements
* Fortran II (1958) : subroutine, function and end
* Fortran III (1958), IV (1962)
* Fortran 66 - 1966, when new ASA (now ANSI) standard was published
  * do loops, data statement, GOTO statement
* Fortran 77
  * block if statements (if-elseif-else), character data type, implicit none, etc.

History of Fortran (3)

- **Fortran 90** - major revision
  - released as an ANSI standard in 1992
  - free-form source input, modules, recursive procedures, derived/abstract data types, dynamic memory allocation, pointers, case construct, and much much more!!!
  - inline comments
  - identifiers up to 31 characters in length
  - new and enhanced intrinsic procedures

History of Fortran (3)

* Fortran 95
  * FOR ALL and nested WHERE constructs
* Fortran 2003 - most recent standard
  * derived type enhancements, object-oriented programming support, asynchronous data transfer
* Fortran 2008 - underway
  * co-array fortran (parallel processing model), BIT data type

Where do you fit?

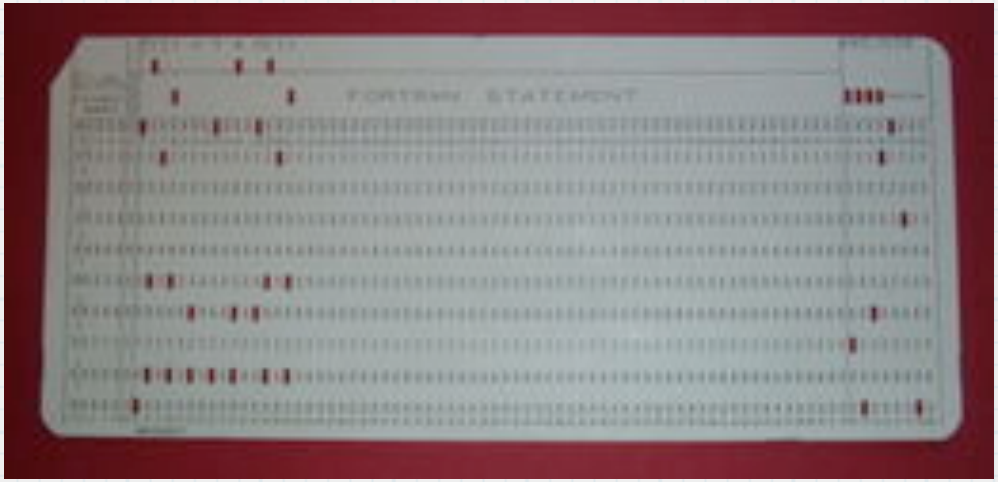
- Most if not all of you fall into one of these two categories:
  - Little to no experience with Fortran.
  - Have programmed with Fortran 77 for years, but haven't **really** learned Fortran 90.
- Try to address both groups.

Motivation

- What's **wrong** with Fortran77?
  - No user-defined data types or data structures (except the COMMON block).
  - Too easy to make mistakes which the compiler could not detect, especially when calling subroutines.
    - Study of more than 4 million lines of professional Fortran showed that **17%** of procedure interfaces were defective.
  - Poor control structures made it hard to avoid using GOTOs and labels.

Motivation (2)

- Archaic features left over from the punch-card era:
  - fixed-format lines
  - statements all in upper-case
  - variable names limited to 6 characters



```
      DO 5005 I = 1,np2jm2
        TEMV(I,1)= CP * BPS(I,1) * VENTFC(I,1)
        FHS(I,1)   = FSS(I,1)       + HLTM * FWS(I,1)
        FSVS(I,1)  = TEMV(I,1)   * THVGM(I,1)
        PSBLOC(I,1)=PS(I,1)-PB(I,1)
       PBBPSK(I,1)=BPB(I,1)/BPS(I,1)
        TS(I,1)    =HM(I,1)-GRAV*ZS(I,1)
       TS(I,1)    =(TS(I,1)-HLTM*WM(I,1))*CPINV
        ZB(I,1)    =ZS(I,1)+CPBG*TS(I,1)*(1. e 0-PBBPSK(I,1))
 5005 CONTINUE
C
      CALL VAMAX(1. e 0,1,SPEEDM,np2jm2)
      CALL R8BTGT(PC,np2jm2,PB,np2jm2,STRTS)
C
      DO 5006 I = 1,np2jm2
        TB(I,1)=TS(I,1)*PBBPSK(I,1)
 5006 CONTINUE
```

Our First Program

- temperature conversion program
- two distinct areas:
  - **specification part** - declare all variables
  - **execution part** - reads in data, calculates new temperature values, and writes them out.
- **ALWAYS** use "implicit none". This means that all variables must be declared! **Bottom line**: It helps the compiler find your errors!
- other variable types: integer, character, logical and complex

Basic Format and Syntax

* Fortran is **case-insensitive**.
* **Symbolic names** can be up to 31 characters long, and may include underscores as well as digits.

```
temperature_in_fahrenheit = temperature_in_celsius * 1.8 + 32.0
waveFunction = pointNum * basisFuncNum ! mixed case works well, too.
```

* **Semi-colons** can be used to separate two or more statements on the same line.

```
sumx = 0.0; sumy = 0.0; sumz = 0.0
```

* **End-of-line comments** start with an exclamation mark (but must not be in column 6 of fixed-format code).

* **Character constants** may be enclosed either in a pair of apostrophes or double-quotes:

```
write(*,*) 'Pass me that chocolate donut.'

write(*,*) "If it ain't broke, don't fix it"
```

* **Relational operators** may be given in old or new forms.

```
old form:  .GE. .GT. .EQ. .NE. .LE. .LT.

new form:  >=    >    ==   /=    <=   <
```

Free-format layout

* Most compilers assume **free-format** if the source file has an extension of .f90 and **fixed-format** otherwise.
  * Can usually override with -free and -fixed switches.
* **Statements** can appear anywhere on a line, and lines may be up to 132 characters long.
* **Comments** start with an exclamation mark "!"
* **To continue a statement** put an ampersand at the end of each incomplete line:

```
call predict (mercury, venus, earth, &  !  comment allowed here
              mars, jupiter, saturn, uranus, neptune, pluto)
```

* If the line-break splits a name or constant, then a comment is not allowed, and the next line must start with another ampersand:

```
write(*,*) "Colorado State University, Department of &
          &Atmospheric Science"   ! NO comment on preceding line
```

* **Spaces are significant** in free-format code: embedded spaces are not allowed in variable names or constants, but a space is generally required between two successive words

```
million = 1 000 000     ! valid in fixed-layout lines only
```

* **Indentation** makes code much easier to read! There's no hard and fast rules about indentation, but indenting by 2-5 spaces, or with a tab, is good practice.

How to write a computer program

* There are four main steps:
  1. Specify the problem
  2. Analyze and break down into a series of steps toward solution
  3. Write the fortran 90 code
  4. Compile and run.
* It may be necessary to iterate between steps 3 and 4 to remove any mistakes. This testing phase is very important.

A Quadratic Equation Solver: The Algorithm

* The problem: Write a program to calculate the roots of a quadratic equation of the form:

$$ax^2 + bx + c = 0$$

* The roots are given by the following formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

* The algorithm

1. Read values of a, b and c.
2. If a is zero, then stop as we do not have a quadratic.
3. Calculate the value of the discriminant $D = b^2 - 4ac$
4. If D is zero than there is one real root: $\frac{-b}{2a}$
5. If D is greater than zero, than there are two real roots: $\left(-b + \sqrt{D}\right) / 2a \qquad \left(-b - \sqrt{D}\right) / 2a$
6. If D is less than zero, than there are two complex roots: $\left(-b + i\sqrt{-D}\right) / 2a \qquad \left(-b - i\sqrt{-D}\right) / 2a$
7. Print solution

Table 2. Statement Order

<table>
<tr><td colspan="2">(1)PROGRAM, FUNCTION, SUBROUTINE, MODULE, or BLOCK DATA Statement</td></tr>
<tr><td colspan="2">(2)USE Statements</td></tr>
<tr><td rowspan="2">(3)DATA, FORMAT, and ENTRY Statements</td><td>(4)Derived-Type Definitions, Interface Blocks, Type Declaration Statements, Specification Statements, IMPLICIT Statements, and PARAMETER Statements</td></tr>
<tr><td>(5)Executable constructs</td></tr>
<tr><td colspan="2">(6)CONTAINS Statement</td></tr>
<tr><td colspan="2">(7)Internal Subprograms or Module Subprograms</td></tr>
<tr><td colspan="2">(8)END Statement</td></tr>
<tr><td colspan="2">Statement Order

Vertical lines delineate varieties of statements that can be interspersed, while horizontal lines delineate varieties of statements that cannot be interspersed. The numbers in the diagram reappear later in the document to identify groups of statements that are allowed in particular contexts. A reference back to this section is included in the places where these numbers are used in the rest of this document.</td></tr>
</table>

Intrinsic Types

* Fortran 90 has **three** broad classes of object type:
  1. **character**
  2. boolean: **logical**
  3. numeric: **integer**, **real**, **complex**
* Notes:
  * there are only two logical values (.true. and .false.)
  * reals contain a decimal point, integers do not.
  * there is only a finite range of values that numeric values can take

Numeric and Logical Declarations

* A simplified syntax for declarations is:

  <type> [,<attribute list>] :: <variable list> [=<value]

```
real :: x

integer :: i, j

logical :: am_i_hungry

real, dimension(10,10) :: y, z

integer :: k = 4

character :: name

character(len=32) :: str
```

Constants (Parameters)

* Symbolic constants (called parameters in Fortran) can be set up with an attributed declaration or a parameter statement

```
real, parameter :: pi = 3.14159
```

OR

```
real :: pi
parameter :: pi = 3.14159
```

* Character constants can assume their length from the associated literal (LEN=*):

```
character (len=*), parameter :: son='bart', dad="Homer"
```

- Parameters should be used:
  - If it is known that a variable will only take one value
  - For legibility where a "magic value" occurs in a program such a pi
  - For maintainability when a "constant" value could feasibly be changed in the future.

```
real, parameter :: grav=9.81, gravi = 1.0/grav, &
                    gas_const_R = 287., &
                   spec_heat_cp = 1005., &
                   hltm = 2.52E+06, &
                    ...
```

Initialization

* Variables can be given initial values using initialization expressions, but these may only contain **PARAMETERS** or literal constants:

```
real :: x, y = 1.0E5
integer :: i = 5, j =100
character(len=5) :: light='Amber'
character(len=9) :: gumboot = 'Wellie'  ! will be padded to the right with blanks
logical :: on = .TRUE., off = .FALSE.
real, parameter :: pi = 3.14159
real, parameter :: radius = 3.5
real :: circum = 2 * pi * radius
```

* In general, intrinsic functions **cannot** be used in initialization expressions, although some can be (e.g., RESHAPE, LEN, SIZE, HUGE, TINY, etc.).

Expressions

* The basic component of an expression is a primary. Primaries are combined with operations and grouped with parenthesis to indicate how values are computed. Examples:

```
5.7e43                  ! constant
number_of_bananas       ! variable
f(x,y)                  ! function value
(a+3)                   ! expression enclosed in parenthesis
```

* More complicated expressions: usually involve the basic form operand operator operand

```
x + y          or          -a + d * e + b ** c
“Ward” // “Cleaver”    or    x // y // “abcde”
la .and. lb .eqv. .not. lc
```

* Each of the three broad type classes has its own set of intrinsic (built-in) operators, like +, //, and .AND.

Assignment

* Assignment is defined between all expressions of the same type. Examples:

```
a = b
c = SIN(.7)*12.7
name = initials // surname
bool = (a == b .OR. c /= d)
```

* The LHS is an **object** and the RHS is an **expression**.