

Lecture V: Introduction to Programming Using Fortran

Charles B. Moss ¹

¹University of Florida

August 30, 2012

1 Web Resources

2 Simple Fortran Program

- Cards and Columns
- Utility Function Fortran Code
- Describing the Lines
- Informative Detour
- Structured Code

Web Resources

Computer programming is one of those topics where learning by doing is the only viable alternative. However, I found two good resources on the Internet:

- <http://www.math.utah.edu/~beebe/software/fortran-documentation/ftnsum.pdf>.
- <http://www-solar.mcs.st-and.ac.uk/~steveb/course/notes/set2.pdf>

Let's start with the simple application from last lecture – Suppose that we want to write a Fortran Program to compute the value of a Cobb-Douglas utility function.

Cards and Columns

- First, Fortran as a specific line format which grew out of the old cards. Each card was 80 characters wide.
 - Column 1 is reserved for a comment statement. Any line beginning with "c" is a comment line that is not executed.
 - Columns 2-5 are reserved for labels.
 - Column 6 is a continuation column. If you need to continue a statement from one line to another, enter a non-zero character in column 6. I usually enter the number of the continuation – The second line I enter a "1" in column 6 to denote the first continuation line.
 - Columns 7-72 are where you write the actual programming statements.
 - Columns 73-80 were historically reserved for card numbers. Don't drop the deck.
- Rumor has it the the most recent implementations have relaxed the strict column conventions, but old habits die hard.

Utility Function Fortran Code

```
program util
implicit real*8(a-h,o-z)
double precision x(3),a(3),g(3),dutil
data x /5.00,5.00,7.50/
data a /0.35,0.25,0.40/
open(unit=6,file="Example01-01.out")
```

C_____

C

C Compute the utility function

C

C_____

```
      dutil=(x(1)**a(1))*(x(2)**a(2))*(x(3)**a(3))
```

C_____

C

C Compute the gradients of the utility function

C

C_____

```
      do 50 i=1,3
```

```
         g(i) = a(i)*dutil/x(i)
```

```
50 continue
```

```
      write(6,100) (x(i),i=1,3),dutil,(g(i),i=1,3)
```

```
100 format(' ',10f10.4)
```

```
      end
```

Describing the Lines

- "program util" The first statement declares the code as a program named util. In more complex codes it is actually possible to pass information into the program. For example, if we had written the command as "program util(x)" then when we executed the code by typing "util 10" the initial value of the variable "x" would be 10.
- Remembering our discussion of variable types - Fortran considers all variables that begin with a-h and o-z to be real numbers (i.e., floating point) and variables that begin with i-n to be integers.
- The statement "implicit real*8(a-h,o-z)" declares that all the real variables are double precision.

Describing the Lines - Continued

- The statement "double precision x(3),a(3),g(3),dutil" does a couple of things.
 - First, I am declaring x, a, and g as "arrays" or tables of 3 numbers. In general I could assign them to be x(n1), a(n2), and g(n3) as long as n1, n2, and n3 are integers.
 - Arrays can be of more than one dimension. For example "double precision b(3,4)" defines a two dimensional array.
 - Second, though redundant, I am declaring these variables to be double precision.

Informative Detour

- The "data x /5.00,5.00,7.50/" statement is an efficient way to put data into an array.
 - As an informative detour, consider a slightly different program based on the multidimensional array "b(3,4)"

```
program simple
implicit real*8(a-h,o-z)
double precision b(3,4)
data b /1.00,2.00,3.00,
1    4.00,5.00,6.00,
2    7.00,8.00,9.00,
3    10.00,11.00,12.00/
open(unit=6,file="Example01-03.out")
do 50 i=1,3
    write(6,60) (b(i,j),j=1,4)
50 continue
60 format(' ',10f10.4)
end
```

Program Output

1.0000	4.0000	7.0000	10.0000
2.0000	5.0000	8.0000	11.0000
3.0000	6.0000	9.0000	12.0000

Describing the Lines - Continued

- `"open(unit=6,file="Example01-01.out")"` declares an output or input file. Here we will use it as an output file.
- `"dutil = (x(1)**a(1))*(x(2)**a(2))*(x(3)**a(3))"` actually computes the value of the utility function.
 - Following standard notation `"+"` is addition, `"-"` is subtraction, `"*"` is multiplication, `"/"` is division, and `"**"` is raised to the power.
 - Ordering or precedence in operation is very important: Power takes precedence over multiplication and division which takes precedence over addition and subtraction. **use parenthesis liberally.**

Describing the Lines - Continued

- The next series a statements is a loop – A sequence of code which will be executed numerous times.
 - "do 50 i=1,3" says to do the lines between here and the "labelled line" 50 ("50 continue") for $i = 1$ to 3 times.
 - Again note that "i" is an integer.
- "write(6,100) (x(i),i=1,3),dutil,(g(i),i=1,3)" is an output statement. It writes the results to unit=6 using the format at the line labeled "100".

5.0000 5.0000 7.5000 5.8804 0.4116 0.2940 0.3136

Structured Code

- An important dimension of writing computer code is the structure of the program.
 - Notice in the code, I start by declaring variables and initiating values.
 - I then do my mathematical operations.
 - Finally I include my format states.
- Structure makes writing code more systematic and errors easier to diagnose.
- An additional point of structure is the subroutine.

```

program simple
implicit real*8(a-h,o-z)
double precision a(3,4),b(3,4),c(3,4)
data a /12.00,11.00,10.00,
1      9.00, 8.00, 7.00,
2      6.00, 5.00, 4.00,
3      3.00, 2.00, 1.00/
data b /1.00,2.00,3.00,
1      4.00,5.00,6.00,
2      7.00,8.00,9.00,
3      10.00,11.00,12.00/
open(unit=6,file="Example01-04.out")
n1 = 3
n2 = 4
call addmat(a,b,c,n1,n2)
do 50 i=1,3
    write(6,100) (a(i,j),j=1,4)
50 continue
do 60 i=1,3
    write(6,100) (b(i,j),j=1,4)
60 continue
do 70 i=1,3
    write(6,100) (a(i,j),j=1,4)
70 continue
100 format(' ',10f10.4)
end

```

```
subroutine addmat(a,b,c,n1,n2)
double precision a(n1,n2),b(n1,n2),c(n1,n2)
do 20 i=1,n1
    do 10 j=1,n2
        c(i,j) = a(i,j) + b(i,j)
10    continue
20 continue
end
```

Output

12.0000	9.0000	6.0000	3.0000
11.0000	8.0000	5.0000	2.0000
10.0000	7.0000	4.0000	1.0000
1.0000	4.0000	7.0000	10.0000
2.0000	5.0000	8.0000	11.0000
3.0000	6.0000	9.0000	12.0000
13.0000	13.0000	13.0000	13.0000
13.0000	13.0000	13.0000	13.0000
13.0000	13.0000	13.0000	13.0000