12.010 Computational Methods of Scientific Programming

Lecturers
Thomas A Herring, Room 54-820A, tah@mit.edu
Chris Hill, Room 54-1511, cnh@gulf.mit.edu

Web page http://www-gpsg.mit.edu/~tah/12.010
Review of Lecture 4

- Looked at Fortran commands in more detail
  - Looked at variables and constants
  - IO commands:
    - Open/Close
    - Read/Write
    - Format
  - Started looking at character strings
Today’s Class

• More Fortran Details
  – Character strings
  – Control statements
    • IF statements
    • DO statements
  – Other command types
    • Include
    • Common
    • Parameters
    • Data statement
Character strings

- Character strings in Fortran are the only variable types where you can easily use pieces of it. (Fortran90 allows character type manipulations with other types of variables)
- To address part of string use : separator i.e., name(n:m) means characters n through m of name
- A single character in a string is referred to as name(n:n) Note: name(n) means the n'th string of an array of character strings.
- For all other array types in fortran there is no easy way to refer to a part of an array. You can pass the whole array to a subroutine, or part of an array by passing starting a some location in the array, or a single element with array(i,j).
- Character strings can be concatenated with the // operator
- Fortran “knows” about the lengths of strings so larger strings can be equated to smaller strings (copy is truncated) or smaller strings copied to larger strings (copy is padded with blanks).
- When strings passed to subroutines or functions, best to declare as character(*) in subroutine or function.
Fortran is (generally) case insensitive and so the auto capitalization of commands here is not a problem. In the other languages it can be problem.
If statements 02

- A single executable statement can be included after the if (no then used) e.g.
  ```
  if( a.eq.b ) c = d
  ```
- When logical variables are used then only the logical need appear
  ```
  logical done
  ...
  if ( done ) then
      Print *, ' Program is finished'
  end if
  ```
- Code within an if statements should be indented with the indentation increasing with each layer of if statements.
Control statements DO

- The basic do construction in Fortran comes in several forms. The two main forms are:
  
  do j = start, end, inc
  <-
  end do
  do while ( logical expression)
  <-
  end do

- In the first form, j starts at value start and continues incrementing by inc until j is greater than end. If inc = 1 then it does not need to appear.

- Often a numeric label is put after the do; this label is then used to end the loop (label in columns 1-5, statement at end should be continue which is no-operation statement.

- If end < start and inc is positive, then the code inside the loop is never executed (-onetrip option is available for fortran 66 compatibility)

The conventional do loop in fortran looks like (note spacing)

do 100 i = 1, 1, 20
  Statements
  100  continue
Control DO 02

- Optionally a numeric label can appear after the do in the first form and the loop will end at the (non-format) statement with that label.
- This form is not recommended, but if used the labeled statement should be a continue.
- j can be real or integer, but for machine independent results, integer variables are recommended.
- The code inside the do loop should be indented.
- Loops may be nested, but may not overlap in range (latter is not possible with recommended form).
- The index variable in the loop should never be modified nor its value used outside of the loop.
Other command types

- include — allows a block of code to be included in the source file. Usually this code is declarations of variables for a common block (see below) or a set of parameter statements.
  
  ```
  include 'file name'
  ```

- Common — allows the declaration of variables that are available in all modules without them being explicitly passed into the module. This declaration should be used with the include statement. Example:
  
  ```
  Real*8 a, b, c, d(10)
  common / reals / a, b, c, d
  ```

- If placed in its own file, all modules that need any of the variables should have the file included.
Common blocks

- The label between the //s names the common and is an arbitrary label.
- Strict Fortran: Only one type of variable should be placed in a common block (not strongly enforced). Different labeled commons can be used for each variable type.
- Many computer scientists do not like commons because any module with the common included can change the value of the variable. However, by use of include, it is easy with grep to find all uses of the variables in a common.
- Variables in commons can be passed into modules but if the common is included in the module, the name needs to be changed.
Parameters

- The parameter statement is a way of naming constants. Again very useful when the include statement is used. Example:
  
  ```fortran
  Real*8 pi, rad_to_deg
  Parameter ( pi = 3.1415926535897932d0 )
  Parameter ( rad_to_deg = 180.d0/pi )
  ```

- Notice that the parameters themselves can be included in other parameter statements.
- Parameters are only available in modules in which they have been declared (thus the use of include statements)
- Parameters can be used to set the dimensions of variable arrays.
Data Statement

- Data statements are used to initialize variables. Strictly, variables initialized in data statements should not be changed by any module (however they can be changed).
- Format of a data statement is:
  
  ```plaintext
  Integer*4 days_in_month(12) ! Day of year number at
  ! start of each month
  ! (Valid in non-leap year)
  Data days_in_month / 0, 31, 59, 90, 120, 151,
  ! 181, 212, 243, 273, 304, 334 /
  .
  ```

- Variables in common can only be in data statements in a module type called block data.
- Exact number and type of values must appear in data statement.
Save statement

• You should assume that variables local to a module will have arbitrary values each time the module is called.
• If you want variables to retain their values, then either use a data statement (implying their value will not change) or use a save statement. Example
  
  ```
  Real*8 last_count
  Save last_count
  ```
• Each time the module called, last_count will retain the value that it had at the end of the last call. The value the first time the module is called is not known, so it should be initialized in the first call.
Exercises using FORTRAN

- Remainder of class we develop and debug fortran programs. Programs will be written in class to
  - Print “Hello World”
  - Compute root-mean-square scatter (RMS) of random numbers generated with the intrinsic rand function
- Students with laptops might want to bring them along so that can work on their own system or on athena. There is wireless internet in the room.
poly_area.f and vars.f should be stored in text files. If they are displayed in a web browser window and the page saved, they should appear as text files on your system. The reason that the files sometimes do not display correctly is the way that the “end-of-line” is denoted on different OS types. In Unix, a new-line character (ASCII Octal 12 or ^J) is used, on Windows a carriage return/new-line and on MacOSX, a simple carriage return is used (ASCII octal 15, ^M).
Exercises using FORTRAN

• In this exercise session we will write some simple FORTRAN programs:
  − Write a simple program that writes your name to the screen
  − Compile and load the poly_area.f program from the web page. Test the program to see how it works
  − Compile and run the other programs from the web page.
  − Compile and load the vars.f routine from the web page. Test the following modifications to the program:
    • In the first call to var_sub_01, replace j with an integer constant and see what happens
• To run fortran:
gfortran <options> <source files> -o <program name>
e.g. gfortran poly_area.f -o poly_area

We will be able to try these programs with g77, and fort77 on Mac OSX plus using gfortran on linerva.mit.edu