12.010 Computational Methods of Scientific Programming

Lecturers
Thomas A Herring
Chris Hill

Review of last lecture

- Start examining the FORTRAN language
- Development of the language
- "Philosophy" of language: Why is FORTRAN still used (other than you can't teach an old dog new tricks)
- Basic structure of its commands
- Communications inside a program and with users
- This lecture will go into commands in more detail
- There are many books on Fortran and an on-line reference manual at: http://www.fortran.com/fortran/F77_std/rjcnf0001.html

Today's Class

- Continue from end of last lecture:
  - Communication
  - Program compiling and layout
- Fortran Details
  - Subroutines and functions
  - Intrinsic routine (e.g., sin, cosine)
  - Constants and variables (plus example)
  - Input/Output
    - Open and close statements
    - Read and write statements
    - Formats
  - Character strings

Communication

- Communications between modules
  - Return from functions
  - Common blocks (specially assigned variables that are available to all modules)
  - Save (ensures modules remember values)
  - Data presets values before execution (during compilation)
  - Parameter (method for setting constants).

Other types of commands

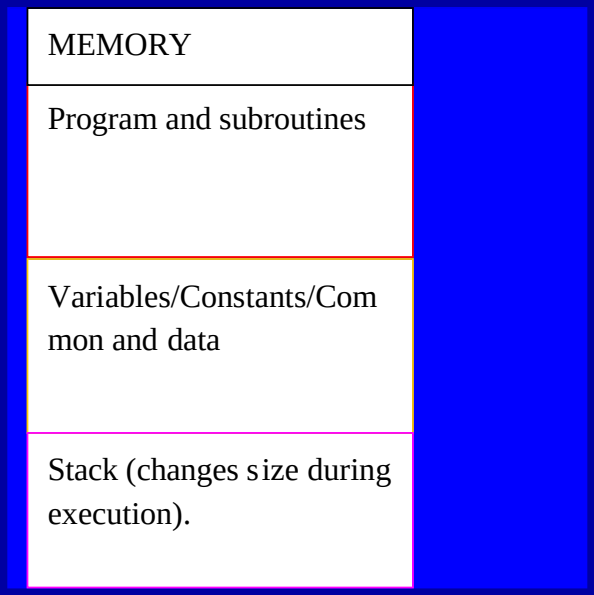
- Other types of commands
  - Open (opens a device for IO)
  - Close (closes a device for IO)
  - Inquire (checks the status of a device)
  - Backspace rewind change position in device, usually a file).
  - External (discuss later)
  - Include (includes a file in source code)
  - Implicit (we will use in only one form.

Syntax

- Relatively rigid (based on punched cards)
  - Commands are in columns 7-72 (option exists for 132 columns but not universal).
  - Labels (numerical only) columns 1-5
  - Column 6 is used for “continuation” symbol for lines longer than 72 characters.
  - Case is ignored in program compilation (but strings are case sensitive i.e., a does not equal A)
  - Spaces are ignored during compilation (can cause strange messages)

Program Layout

- Actual layout of program in memory depends on machine (reason why some “buggy” program will run on one machines but not others).
- Typically executable layout in memory.



- Not all machines use a Stack which is a place memory is temporarily allocated for module variables.
- Good practice to assume stack will be used and that memory is “dirty”

Compiling and linking

- Source code is created in a text editor.
- To compile and link:

  f77 <options> prog.f subr.f libraries.a -o prog

Where prog.f is main program plus maybe functions and subroutines

  subr.f are more subroutines and functions

  libraries.a are indexed libraries of subroutines and functions (see ranlib)

  prog is name of executable program to run.

- <options> depend on specific machine (see man f77 or f77 -help)

Basic f77 options

- Options differ greatly between different machines although there are some common ones (these are not universal)
  - -c compile only do not link
  - -u assume implicit none in all routines
  - -ON where N is level of optimization. Optimization can lead to significant speed increases but on complex codes can generate strange errors.
  - -g compile for debugging
- Typically many more options often to provide for use of old codes (e.g., -onetrip). We will not explore these but useful to check if trying to get someone else's code running.

Basic layout and command details

- A basic Fortran program looks like (see poly_area.f for example).

```
      program name
*     Comments
      Non-executable declarations
         ……
      executable statements

      end
      subroutine sub1
*     Comments
      Non-executable declarations
         ……
      executable statements
      return
      end
```

Character of commands

- Modules are invoked by call for subroutines and assignment statements for functions.
- Certain system level modules are invoked just through their names. For example
  - OPEN Opens a files (takes arguments)
  - CLOSE Closes a file
  - READ and WRITE are of this type
- User modules or routines (these are the building blocks) are of types:
  - SUBROUTINE
  - FUNCTION

Subroutines (declaration)

Subroutine name(list of variables)

- Invoked with

  Call name(same list of variable types)

- Example:

  Subroutine sub1(i,value)
  Integer*4 I
  Real*8 value

In main program or another subroutine/function:

integer*4 j
Real*8 sum
Call sub1(j, sum)

Note: Names of variable do not need to match, just the type needs to match, although it is good practice to do so if possible

Variables used in subroutines (and functions) are in general are lost after the subroutine completes execution. Use the “Save” command if variables values are to be remembered.

Functions

Real*8 function func(list of variables)

- Invoked with

  Result = func(same list of variable types)

- Example

  Real*8 function eval(i,value)
  Integer*4 I
  Real*8 value
  eval = I*value

In main program or subroutine or function

Real*8 result, eval
Integer*4 j
Real*8 sum
Result = eval(j,sum)

Functions 02

- Functions can be of any of the variable types
- The last action of the function is to set its name to the final result
- The function type must be declared in the main program (looks like any other variable)
- There are other forms of the declaration. Often simply function is used and the type declared in the function.
- The function must always appear with the same name and type.
- Fortran has special functions called intrinsic which do not need to be declared.

Intrinsic functions

- These functions are embedded in the language and often go by "generic names"
- Examples include sin, cos, tan, atan2. Precisely which functions are available depend on machine.
- Generic names means that the actual loaded code depends on the variable types used in the call (i.e., if real*8 argument is used in sin, then the real*8 version of the sin function is loaded).
- Link to standard intrinsic functions
- Not all of the intrinsic listed on this page are available always. If not available: Get an undefined external message or undeclared variable when program compiled and linked.

Variables and constants

- In Fortran variable names point to an address in memory and so most of the time when variables are passed only the address is passed. There is usually no check that information about the variable type between the modules.
- Character strings are treated differently since the string is defined not only by an address but also a length. Unlike Fortran arrays, inside a module you can tell the length of string passed (LEN intrinsic function). (Fortran90 does have features that allow the sizes of arrays to be determined: size, shape, lbound,ubound)
- Constants may be passed by address or by value. Passing by value is the more common technique now.
- Variable type Fortran Program

IO: Read, write, open, close

- These are the main routines used to get data into and out of a program.
- Format of the read and write commands are:

Read(unit,format,<options>) list of variables

Write(unit,format,<options>) list of variables

Where unit is either:

A numeric number associated with a device or file. Set with an open statement

* which is generic for screen or keyboard

A character string (call internal reads and writes).

Format is a format statement defining how variables are read/written or a * for free-format.

IO Open

- To open a unit number for IO you use the open command. Its basic format is

  Open(unit=nnn,file=<string>, status=<status>,iostat=<ierr>,…)

Where nnn is a numeric value, e.g., 80

<string> is a string containing the file name. This can be a character variable or the name of the file contained in single quotes e.g., 'prog.dat'

<status> is a string with type options

'unknown' status unknown

'old' file should already exist

'new' file should not exist.

IO Open continued 02

<ierr> is an integer*4 variable for the IO Status (IOSTAT) return. Most important here is that 0 return means the open was successful. Any non-zero return means an error occurred and the file is not open.

The specific numerical values for given types of errors depends on the machine.

The meaning of the numerical values can be found in the "runtime error" or "IOSTAT error" section of the manual for the fortran on your machine. (HP: 908 means file not found, g77 Linux: 2 means the same, Solaris: 1018).

- There are more options for direct access files (fixed length records), read-only, to append, binary files (unformatted)

IO Close

- The close statement closes a unit number.

  Close(unit=<nnn>,iostat=<ierr>)

Where <nnn> is the unit number, and

ierr is IOSTAT error for the close.

- For both open and close, the unit= is not needed and the numeric value of the unit number of all that is needed.
- The unit number can be an integer*4 variable containing the unit number.
- In some cases, the default compilers require the unit number be less than 99.

Open/close

- Some unit number are automatically opened when a program is executed.
- Unit 5 is for reading from keyboard
- Unit 6 is write to screen
- Unit 0 is often (but not always) the error output device.
- You should not close units 5 or 6
- Generically, * can be used in read and write for reading from keyboard and writing to screen. This is the preferred option.
- For reads and write to * or 5 or 6, the Unix IO redirect can be used (< file, > file, or | for piping.)

Read/Write

- For read and write the most common <option> is IOSTAT=ierr which allows the IO error during read/write to be checked.
- If the IOSTAT= option is not included, and an error during reading or writing then your program will “core dump” or “abort”
- When the IOSTAT= option is included, it is your responsibility to check that the return is zero.
- The most common return is -1 mean End-of-file has been reached.
- Other options exist for reading specific records from direct access files

FORMAT

- Allows the format of input or output to specified.
- There are two ways to specify a format:
  - A numeric label can be given which refers to labeled format statement
  - A character string containing the format can be given.
- The following two cases generate the same result.

```
    Write(*,100) I
100 format( ‘ The integer value is  ‘,i4)
    Form =  ‘(” The integer value is “,i4)’
    Write(*,form) I
```

- Note the use of single and double quotes

FORMAT arguments

- The options in the format statements are:
  - Strings inside single or double quotes are written as is.
  - For numbers the generic type is <T><n>.<m>

  Where T is for type; I integer, F floating point notation, E exponential, L logical, A for character strings, G for F and E combination !!

  <n> is total width of field

  <m> is decimal places or leading zeros.

  Examples: if 22.7 is written with:

  F8.3 ^^ 22.700 (with spaces (^) 8 characters wide, 3 DP)

  E11.3 ^^ 0.227E+02

  For integer 10 I4.3 -> ^ 010 (The symbol ^ means space)

FORMAT 02

- Character strings are output with the a format

A10 would write 10 characters, left justified (if the string to printed is longer than this it is truncated).

A without any numeric value following it will print the full declared length of the string.

- Other control characters are:

/ — carriage return, start a new line

Nx — print N spaces where N is integer

Enclosing part of the format in parentheses with with a numeric argument in front, repeated that parts of format N times, e.g., 20(I4,2x)

Character strings

- Character strings are treated differently in Fortran than other variable types because an address and length is passed with the name.
- In side modules: Strings that are passed in can be declared as

  Character*(*) <string name>

Where the second * says use the “passed length” of the string.

(Homework number 2 which will be due Oct 20 will use these concepts).

Summary of Today's class

- Fortran Details
  - Subroutines and functions
  - Intrinsic routine (e.g., sin, cosine)
  - Constants and variables (plus example)
  - Input/Output
    - Open and close statements
    - Read and write statements
    - Formats
  - Character strings
- For the remainder of the class; examine, compile and run the poly_area.f and test programs: loops.f, ifs.f, inout.f and subs.f
- vars.f is a special examples program.
- Try modifications of these programs and see what happens.

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

12.010 Computational Methods of Scientific Programming
Fall 2011