Review of Lecture 2

- Examined computer hardware
- Computer basics and the main features of programs
- Program design
- Examined some of the pitfalls in programming:
  - addition of numbers,
  - rounding error in functions
  - Mixed number computations
  - Expected units.
Today’s 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
• Next lecture will go into commands in more detail
• There are many books on Fortran and an on-line reference manual at:
  http://www.pa.msu.edu/~donev/FORTRAN77/

FORTRAN (Formula Translation).

• History
  – Developed between 1954-1957 at IBM
  – FORTRAN II released in 1958
  – FORTRAN IV released in 1962 (standard for next 15 years)
  – FORTRAN 66 ANSI standard (basically FORTRAN IV).
  – FORTRAN 77 standard in 1977
  – Fortran 90 Added new features, in 1997 Fortran 95 released (started phasing out some FORTRAN 77 features).
FORTRAN Philosophy

• FORTRAN developed a time when computers needed to do numerical calculations.
• Its design is based on the idea of having modules (subroutines and functions) that do calculations with variables (that contain numeric values) and to return the results of those calculations.
• FORTRAN problems are a series of modules that do calculations with typically the results of one module passed to the next.
• Usually programs need some type of IO to get values to computations with and to return the results.

FORTRAN 77:

• Commands are divided into executable and non-executable ones. All non-executable commands must appear before the executable ones.
• Syntax is quite rigid (discuss later)
• Basic Structure:
  – Module types:
    • Program (only one per program, optional)
    • Subroutine — Multi-argument return
    • Function — single return (although not forced)
    • Block data — discuss later
Basic Structure 02

– Variable types
  • Integer\*n where n is number of bytes (2/4)
  • Real\*n where n is 4 or 8
  • Complex\*n where n is 4 or 8
  • Character\*(m) where m is number of characters in string variable
  • Logical\*n is n is 2 or 4
  • Byte (equivalent to integer\*1)

– In this course, all variables must be explicitly declared. In Fortran all variable types must be declared implicitly or explicitly before executable statements.

Basic Structure 03

– Constants: Numerical, strings or defined in parameter statement

– I/O
  • Read (various forms of this command)
  • Write (again various forms)
  • Print (useful for debug output)
  • Command line arguments (discuss more later)
  • Format defines how results are read and written.

– Math symbols: \* / + - ** (power) = (assignment). Operations in parentheses are executed first, then **. \* and / have equal precedence (left to right), + - equal precedence left to right.
Basic Structure 04

- Control
  - If statement (various forms)
  - Do statement (looping control, various forms)
  - Goto (you will not use in this course)
- Termination
  - End (appears at the end of every module)
  - Return (usually at end of modules, except program)
- Communication between modules
  - Variables passed in module calls. Two forms:
    - Pass by address (memory address of variable is passed)
    - Pass by value (actual value passed, rarer and usually on applies when constants or expressions to be evaluated are passed)

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.

<table>
<thead>
<tr>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program and subroutines</td>
</tr>
<tr>
<td>Variables/Constants/Common and data</td>
</tr>
<tr>
<td>Stack (changes size during execution)</td>
</tr>
</tbody>
</table>

- 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 no 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., -ontrip). We will not explore these but useful to check if trying to get someone’s code running.

Basic layout and command details

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

  ```fortran
  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

Summary

• Look at the basic characteristics of FORTRAN
• Next lecture we look in more details of the syntax of the commands (e.g., the arguments that can be used with an open statement)
• Trial FORTRAN programs will also be shown to examine the actual structure of a program.