FORTRAN Short Course

Week 1

Kate T-C

August 25, 2009
What are we talking about this week?

- Quick Intro to the Class
- Three Laws of Computer Science
- How does a computer actually work?
- Overview of Operating Systems
- Developing a plan and a program
  - Dive into Fortran!
- Variables, Types, Arrays, Arithmetic Functions, Order of Operations
## Introduction to Programming: FORTRAN Short Course

**Scheduling:** Tuesdays 3:00-5:00p in ATS 101  
**Instructor:** Kate Thayer-Calder  
ATS 404  
katetc@atmos.colostate.edu  
**Office Hours every Thurs 2:00-3:30p**  
**Class Website:** [http://www.atmos.colostate.edu/gradprog/programming/](http://www.atmos.colostate.edu/gradprog/programming/)

<table>
<thead>
<tr>
<th>Date</th>
<th>Topics</th>
</tr>
</thead>
</table>
| 25 - Aug | • Another Intro to Computer Science  
• How a Computer Actually Works  
• UNIX Introduction: What is an Operating System?  
• UNIX vs LINUX vs MacOSX vs Windows (basics)  
• Getting ready to program: Compiler, Home directory, Text Editor  
• Creating and Compiling a Fortran Program  
• Print is your friend  
• Variables, Types, Arrays, Arithmetic Functions, Order of Operations |
| 01 - Sep | • Everyday UNIX: Man pages, File and directory manipulation, program input and output, processes, basics of Emacs and Vi  
• More on FORTRAN Arrays  
• Logical Operators  
• Conditional Statements  
• Looping options  
• Subroutines and Functions  
• Discussion of Scientific Computing |
| 08 - Sep | • Using UNIX on a network: Basics of networking, FTP, SSH, Tar/Zip, Mount, network utilities, a little HTML  
• Makefiles  
• Strings, strings, strings  
• Higher dimensional arrays  
• Reading/Writing Formatted and Unformatted Data  
• Basic discussion of modeling |
| 15 - Sep | • Searching on UNIX: Grep, awk, pattern matching and regular expressions  
• Some plotting and visualization options  
• Fast math in FORTRAN  
• Modules and organizing big programs  
• User Defined Data Types  
• Reading/writing scientific data  
• Dealing with missing or poor quality data |
Goals of this Class

- Learn how to write workable, understandable, debuggable Fortran 90/95 (2000) code and programs
- Learn a bit more about how computers work
- Learn how to think about your problem in a way that will help you write good programs
NOT Goals of This Class

- Learn everything there is to know about programming!!!(Every language is different)
- Learn how to read/write Fortran 77 or old Fortran code
- Become an expert in Fortran or HPC
- Learn every option of aspects of Fortran
- Learn about the kinds of programming that could actually make you money
- No PHP, no AJAX, no Java, no OOP at all
THREE LAWS OF
COMPUTER SCIENCE

Because Turing says...
Law #1

The word **CODE** is already plural.

- Think of it like deer or fish

- Corollary: It is only acceptable to say “codes” if you are talking about launching missiles or cryptography.

- And I mean it!
Law #2

Computers ONLY do what you tell them to!

- If something is wrong, it’s probably your own fault. I’m sorry. But it is.

- Corollary: Sometimes you don’t know you told the computer to do it wrong, or somebody else did the telling.
Law #3

Do not re-invent the wheel.

Corollary: You probably won’t know it’s a wheel as you’re inventing it.
Programming Philosophy

- How do you think about computers?
  1. Magic Incantations
  2. Working a machine

- Both are A-ok! But I’m a #2-er, usually.
How does a computer work?
How does a computer work?

CPU
RAM
Hard Drive
Video
BIOS
Power
USB
How does a computer work?

Input → Processing → Output

User → Processor → Visual
Memory → Processor → Memory
Memory → Processor → Visual
User → Processor → Memory

Repeat as necessary!
How does a computer work?

- User Input: Keyboard, Microphone, video camera, digital camera...
- Visual Output: Monitor, printer, speakers, Braille terminal...
- Memory (slowest to fastest): CDs/DVDs, External Hard Drives, flash drives, Internal Hard Drives, RAM, Cache (L3, L2, L1), Registers
- Processor = CPU (Central Processing Unit) takes numbers (in binary) and does math the result is numbers (in binary)
Basically, Computers

- Can store binary numbers.
- The binary can be interpreted into lots of different types of numbers or even text or graphics.
- Binary is divided up into bits, bytes and words.
- Can perform math on binary numbers.
- Everything that you see or do on a computer boils down to a line of math.
- A program is just a long series of mathematical operations performed on lots and lots of numbers.
Surely there’s more than that...?

Right now, I’m using a computer, and the only 1’s and 0’s I see are the ones I’m typing.

The Operating System (OS) provides a layer between the basic operation and the human.

The OS is what makes a computer usable: organizes files, runs programs, handles network communications.

So important, that the richest man in the world made that money from selling this simple piece of software.
Operating Systems

This layer of software allows us to write programs without directly manipulating hardware (code that runs code).

Almost all computers (desktops, supercomputers, video game consoles, robotic systems, washing machines, cars, phones, iPods, etc) have some form of OS.

What we usually see on our work computers is a Graphical User Interface (GUI) or a Command Line Interface (CLI).

For Windows and Macs, the GUI is part of the OS. On UNIX systems, sometimes the GUI is separate.
Operating Systems

- Programs
- User Interface
- File System & Processes
- Shell
- User

Hardware
- External Devices
- CPU / Main Board
- Physical Memory

The Internet

Tuesday, August 25, 2009
Operating Systems

- **Windows** - Real world usage statistics:
  - XP ~70%, Vista ~17%, W7 ~2%
  - ATS Campus: XP ~15%, Vista ~0%

- **Mac OSX** - Real world: ~6%, ATS: ~40%

- **UNIX/LINUX** - Real world: ~5%, ATS: ~45%
Windows

Originally based on MS-DOS, Windows 1.0 was released in 1983.

One of the early GUIs that allowed some form of cooperative multi-tasking.

Shipped with apps such as: Calculator, cardfile, clock, control panel, Paint, Reversi, Terminal

Windows 3.1 (3.1x, 3.11) 1990-1995, improved GUI and memory usage.

Windows 95, 98, Me supported long filenames and included a web browser (uh oh)

Windows NT, a more powerful (expensive) OS for servers, better security, better reliability.
Windows

Tuesday, August 25, 2009
Windows

- Windows NT was considered a superior OS, fewer problems with hardware conflicts, better security, etc.
- Starting with Windows 2000, home versions of the OS were based on the NT framework.
- Windows XP, released in 2001, is still commonly in use.
- Made improvements in efficiency and design for software developers, so it's nearly everywhere (even ATMs, flight or postal kiosks, etc).
- Recent releases include Windows Vista (2006) which was considered a flop. And Windows 7 (Oct 2009?) hopeful redemption!
Problems with Windows

Microsoft wrote an OS that was supposed to run ANY piece of hardware!

All a hardware developer had to do was write a driver that would tell Windows how to use the device.

Sometimes the hardware would do something bad (access restricted memory or resources) and the whole computer would die.

Everybody (92% of the world) uses Windows, which makes it a target for hackers.

In attempting to fix security problems, Windows sometimes becomes annoying.
Problems with Windows

A problem has been detected and windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c


Tuesday, August 25, 2009
Mac OS
Original Mac OS (1984–2001) was written by Apple

Because the OS was stored on a chip and not the hard drive, only certain specific computers could run the operating system

Usually separated into two pieces “System” or the guts and “Finder” which had the pretty UI.

No CLI at all, supposedly the most “user-friendly” OS

Each piece updated separately, but usually near each other, and named numerically (ie: Mac OS 7, 8, 9)
Starting with Version 10, Apple replaced “System” software with a version of Unix called Darwin.

Now, Mac is effectively a pretty face on a Unix body (standing on Apple hardware).

Modern Macs have a CLI through terminal applications.

Apple has maintained a strangle-hold on their operating system, software and hardware, even with more open architecture.
Max OS X
Problems with Mac OSX

- Proprietary hardware and software make the system better tested and stable, but with fewer options
  - Many software packages/hardware toys aren’t available on Macs (especially games)
- Less competition in the area makes Macs more expensive
- In an effort to make the system more user-friendly, some low-level tasks become very difficult and complex
  - If it doesn’t “just work,” you’re probably screwed.
Unix

The oldest OS we’re discussing – was developed at Bell Labs in 1969.

One of the first OSs designed to be portable (usable on any machine), multi-tasking and allow multiple users.

Both Unix and C were distributed freely to academic and government institutions in the early days, so has maintained a historically important place in scientific computing.

Hundreds of variants exist today, most are technically only “Unix-like” but are referred to as “Unix systems”
Unix

Basically, the Unix OS has three parts: The Kernel, The File System and The Utilities.

The Kernel controls all the low-level tasks (memory management, resource allocation, etc).

The File System organizes files (in Unix, devices and drives are treated as files, from which data can be read and written).

The Utilities are the hundreds of little commands and programs that we will use to interact with the computer.
Unix was invented before the idea of a GUI.

Traditionally, any UI is considered a separate layer, and should be interchangeable on the OS and hardware.

We use Unix from a CLI, typically.

We won’t spend much time talking about Unix GUIs in this class.
Unix GUIs?

NAME
xset - user preference utility for X

SYNOPSIS
xset [-display display] [-b] [b [volume [pitch [duration]]
[[-bc] [-c] [c on/off] [c [volume]] [[+-]dpas] [dpas standby [suspend
[off]]] [dpas force standby/suspend/off/on] [[+-]f[p]---
path path\ldots [/]] [fp default] [fp rehash] [[-]led [integer]] [led
on/off] [m[ouses] [accel_mult /[accel_div] [threshold]]] [m[ouse]
default] [p pixel color] [[-]r [keycode]] [r on/off] [r rate delay
[rate]] [s [length [period]]] [s blank/oblank] [s expose/noexpose] [s
on/off] [s default] [s activate] [s reset] [q]

DESCRIPTION
This program is used to set various user preference options of the dis-
play.

OPTIONS

-display display
This option specifies the server to use; see X(7).

b
The b option controls bell volume, pitch and duration. This
option accepts up to three numerical parameters, a preceding
dash(-), or a 'on/off' flag. If no parameters are given, or
the 'on' flag is used, the system defaults will be used. If
the dash or 'off' are given, the bell will be turned off. If
only one numerical parameter is given, the bell volume will
be set to that value, as a percentage of its maximum. Likewise,
the second numerical parameter specifies the bell pitch, in
hertz, and the third numerical parameter specifies the duration
in milliseconds. Note that not all hardware can vary the bell
characteristics. The X server will set the characteristics of
the bell as closely as it can to the user's specifications.

bc
The bc option controls bug compatibility mode in the server, if
Problems with Unix

- Reliance on the CLI and hundreds of little utilities means using the system for simple tasks is less than intuitive.

- In theory, the OS should run on and support any hardware. In practice, most hardware makers only support Windows. (But who needs a mouse anyway?)

- Unix is not a commercially viable platform for most software development.

- Incredibly powerful and flexible, which means it takes YEARS to really learn how to use it.
Originally released in 1991, Linux is a derivative of Unix that is available freely, as an open-source project.

Open-Source: anybody can access, update, fix, hack, or change the code as wanted.

There are a lot of versions of Linux (Red Hat, Debian, Ubuntu, Fedora, etc), but they are all very similar at core.

Basic Unix-like Kernel, file system, and utilities.
* Rule of Modularity: Write simple parts connected by clean interfaces.
  * Rule of Clarity: Clarity is better than cleverness.
  * Rule of Composition: Design programs to be connected to other programs.
  * Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
  * Rule of Simplicity: Design for simplicity; add complexity only where you must.
  * Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
  * Rule of Transparency: Design for visibility to make inspection and debugging easier.
    * Rule of Robustness: Robustness is the child of transparency and simplicity.
  * Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
    * Rule of Least Surprise: In interface design, always do the least surprising thing.
    * Rule of Silence: When a program has nothing surprising to say, it should say nothing.
      * Rule of Repair: When you must fail, fail noisily and as soon as possible.
  * Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
  * Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
  * Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
    * Rule of Diversity: Distrust all claims for "one true way".
  * Rule of Extensibility: Design for the future, because it will be here sooner than you think.
How do we write a program?

- We need a language that is easier than just writing billions of 1’s and 0’s (Let’s use Fortran!)
- We need a program that can translate our computer language into 1’s and 0’s that the computer understands (called the compiler)
- We need a way to tell the computer to run and store our program (the operating system)

Let’s work from the bottom up...
Getting ready to program

- What you’ll need: A computer running some OS, a Fortran compiler, a place to store your programs, and a place to write them.

- We’ve covered OS’s

- Talk to your Sys Admin about compilers (Absoft, Intel, GNU).

- For now, we’ll store our programs in a directory - ~/code/ATSProgramming/Fortran
Creating a directory

In Unix, you start in your “home” directory.

You can go home at any time by typing `cd ~`

Let’s organize – set up some folders/directories:

- `mkdir code`
- `cd code`
- `mkdir FortranCourse`
- `cd FortranCourse`
- `ls -al`

This is where we’ll put our program files.

You can set up any structure that works for you.

Should be able to browse to the directory in Finder.
Choosing a Text Editor

- Talk to your favorite computer person about what they use (you’ll have somebody to ask questions)
- In Unix - Emacs and VI are biggies
- On Macs - We use BBEdit
- Ask your computer person if you can set up a Fortran “mode” that colors code appropriately
PROGRAM ILikePie

! Written by Kate T-C
! 2.11.09 For the Fortran Short Course
! This program is a bad joke.

real :: pi = 3.141592654
print *, 'I Like', pi

END PROGRAM ILikePie
Things in the program

- Each line is a statement
- The program runs top to bottom, in order
- Program start, program name
- Comment block
- Variable declaration
- Output statement - Print is your friend!
- Program end
Compile the Program

>`f90 ILikePie.f90`

Or replace ‘f90’ with a call to your Fortran compiler (need to talk to your local computer person to find this one out)

We can also do...

>`f90 ILikePie.f90 -o Pie`

This is called Compile-Time.
Run the program

> a.out

... or (if we used -o ) ...

>Pie

... or navigate to the program in your window environment and double-click. You wrote a computer program!

This is called Run-Time.
Code Talker

For added security, after we encrypt the data stream, we send it through our Navajo code talker.

...is he just using Navajo words for "zero" and "one"?

Whoa, hey, keep your voice down!
Let’s do another one!

Who uses Fortran? Well, climate modelers! So, let’s write a climate model...

Energy In = Energy Out

\[ S_o (1 - a) \pi r^2 = \sigma T_e^4 \ 4 \pi r^2 \]

\[ \frac{S_o}{4} (1 - a) = \sigma T_e^4 \]
Translating the model into Fortran

- Variables, Literals and Constants
- Types
- Input, action, output
- Implicit None

See example program ClimateModel.f90
Variable Types

- **real** - a floating point number
- **double precision** - a floating point number using twice the bytes for more accuracy
- **integer** - a straight-up whole number
- **complex** - numbers that include an imaginary component
- **logical** - two values: true or false
- **character** - variable contains text
Arithmetic Operators

Computer languages follow the algebraic Order of Operations

1. Parenthesis ()
2. Exponentiation **
3. Multiplication and Division *, /
4. Addition and Subtraction +, -

Please Excuse My Dear Aunt Sally
Arithmetic Operators

Try it out, what do you get for the following statements?

10 - 3 + 2 * 10

(10 - 3 + 2) * 10

3 ** 2 + 1

3 ** (2 + 1)

3 ** 2 ** 0.5
Reals and Integers

Mixing types in arithmetic can be confusing

If operands are all integers, the result is an integer, otherwise, the result is real.

\[ 8.0/4.0=2.0, \ 8.0/4=2.0, \ 8/4=2 \]

Integer division truncates the result

\[ 10.0/4.0=2.5, \ 10.0/4=2.5, \ 10/4=2 \]

We can save some trouble by casting to make sure everything is the correct type

Because computers use binary, technically, ALL division and multiplication is truncated!
Another Example!

What else do we use Fortran for around here?

It's basically the fastest way to manipulate large amounts of data.

To perform operations on data, we could either declare thousands of "real" variables (one for each data point) or just declare one Array.

Check out example Statistics.f90
Arrays

- Give one name to a series of numbers
- Each element in the array has an Index or Subscript – which must be an integer
- You can declare an array of any type using the ‘dimension’ attribute
- You can fill the array when you declare it using (/x,y,z/) notation or fill it at run-time using input data from other sources (files, stdio, instrumentation, etc)
What did we cover today?

- Three Laws of Computer Science
- How a computer works
- Windows vs Mac OS X vs Unix vs Linux
- Unix commands: `cd`, `mkdir`, `ls`
- Writing a Fortran program - text, compile, run
- Literals, Variables and Constants
- Implicit None and Variable Types
- Arithmetic operators and the Order of Operations
- Arrays

Tuesday, August 25, 2009
Homework

Sure, why not?

Email your code and program output to me if you want feedback

My office is ATSW 212 if you need help