Topics for this week

- All about networking
- Using code from other files
- Makefiles
- Writing to and Reading from files
- User defined data types
- Strings!!
- Recursion
- HTML
Computer network: Any group of interconnected computers.

Built in many different architectures and designs.

Computers can connect over phone lines, microwaves, ethernet, fiber optics, or other cables.

We mostly see: Local area networks, Campus-wide networks, and Wide-Area networks (such as the internet or Teragrid).
Aka The Internet Protocol Suite, this is a group of rules for how computers and devices communicate (esp over the internet)

Each computer is assigned a number, or IP address (192.168.0.12 or 129.82.49.49), data is broken down into millions of packets, each one like a letter with a “to” and “from” address on them.

Packets are routed across the internet through millions of routers, who's only job is to direct packets towards their destination. If one router goes down, traffic can go through any other one.
How to log in to another computer: SSH

- Stands for “Secure Shell” and allows you to log into another computer, and use it from a command line.
- You must have an account on the other computer (and know its internet name) to log in.
  
```bash
  ssh eddy.atmos.colostate.edu -l kate
```
- You’ll log in to your account’s home directory
- From there, you can use the distant computer as if you were sitting in front of it
- Google “X11 Forwarding <Your OS Here>” for XTerm
How to move files between computers: FTP

File Transfer Protocol: This program establishes a connection between two computers, and files can be transferred to the host ("put") or from the host ("get")

ftp katetc@eddy.atmos.colostate.edu

Once logged in, use commands at the prompt to navigate the remote computer (ls, cd, mkdir, pwd, exit)

Put [file], Get[file]
Mput [files/*], MGet [files/*]

ascii - faster mode, works only on text files
binary - a little slower mode, works on all files

GUI ftp applications can make the process easier (Cyberduck on the mac is a nice one)
How to move files between computers:  `scp`

- Stands for Secure Copy - and is another way to transfer files between computers
- Uses SSH logins and encryption to protect both passwords and file contents from network sniffers or snoopers

  ```
  > scp [file(s)] user@othercomputer:destination
  > scp -r user@othercomp:SourceDir localDestDir
  ```

- All directories default to relative to your home directory on the remote computer.
  - So data in an FTP directory might be hard to navigate to.
How do I find out an IP Address?

- `host localhost`

  `host` is a simple command that returns the IP address of a named computer (localhost is usually the name of the local computer).

- `dig <name>`

  A slightly more complex/powerful command that looks up the IP information about a named computer.
Ping!

ping [computername/address]

Sends out a series of packets to the computer specified and tells you if they made it back and how long it took.

Can be very useful in diagnosing connectivity problems (ping to see if you’re online, ping to see if the mail server is online, etc)

Also a common verb among computer people “Just a ping to see if you’re out there!”
Other useful commands

- `ifconfig` - gives information about your computer on the network (including IP address and mac address)

- `traceroute` - lists all of the routers and computers a packet travels between you and a destination
  
  > `traceroute google.com`

- `whois` - gives the ARIN information about an internet domain name.

  > `whois patientfirst.com`
Grouping Files

- tar - stands for ‘tape archive.’ A utility that combines files and directories into one tar file or tarball.

- Tar files must be un-tarred (de-tarred?) in order to view their contents. Tarballs are not compressed.

- `tar -xf file.tar` (extract the files from file.tar)

- `tar -cf file.tar ~/foo/` (place the contents of foo into the tar file file.tar)

- `tar -xvzf file.tar.gz` (extract files, uncompress, verbose mode)

- Can also use `zip` or `unzip` for compressed files.
mount

All of the files in your unix system are part of a big directory tree - starts at the root directory and branches out from there.

Mount attaches the file tree from another source (network drive, disk drive, etc) to a specific place in your directory tree.

Used to have to do this a lot more often, now it is usually done automatically.

> mount /dev/cdrom ~/cd (mount the cd)
> umount /dev/cdrom (unmount the cd)
> mount (list all mounted file systems)
Files

- A way of organizing large amounts of data (1's and 0's of course) inside the computer.

- Files are stored on hard drives or disk drives and the information inside of them can be loaded into data structures in RAM.

- File Input/Output (I/O) is the second slowest thing your program can do. Best to keep it to a minimum.

- When accessing code in another file, the compiler (the Linker, actually) will set up the programs in memory so it’s not too slow.
Files

- **Source files** contain your code. All of your code.

- Code that is compiled into a program is called the **Source Code** for that program.

- A compiled or otherwise available program is contained in an **Executable file**.

- Your **main** function or program is the place where your overall program starts and ends. The highest-level list of execution.

- **Rule of Modularity:** Write simple parts connected by clean interfaces.
Code in Multiple Files

One easy way to make code reusable is to create a “library” file, which contains lots of different useful subroutines and functions.

Each subroutine and function must be self-contained (it can only see its own variables).

Make sure you document each one well with descriptive comments.

Compile the library and your program together at the same time to use the external procedures as you would internal ones.

```
f90 Program.f90 Library.f90 -o Program
```
Example

RelativeHumidity.f90 must be compiled with RH.f90

`f90 RelativeHumidity.f90 RH.f90 -o RelHum`

Not compiling the two together results in a Link error – the linker is looking for the function but can’t find it.

Can compile as many files together as you want.
Including Code

Instead of compiling the two files together, we can include external source files using the `include` statement at the top of our file.

This, effectively, copies and pastes all of the text from the included file into the program file, and the compiler "sees" the functions in the included file as being all in the same file.

This is kind of a compiler hack from the early days, but is the easiest way to add old code to your new code.

See example RelativeHumidity2.f90
Fortran 90/95 Modules

Like a program except they don’t actually *do* anything on their own.

Exist so that ANYTHING required by more than one program unit may be packaged in a module and made available to all that need it.

Example atmosLib.mod

```
MODULE name
  implicit none
  ... (variables, types, other code, etc)
CONTAINS
  ... (subroutines & functions)
END MODULE name
```
Modules

- All of the subroutines and functions and any program that uses a module can see the data in the main part of the module.

- This is a good place for **Global** variables

- To use a module, add the line at the start of your program declaration

  `USE module_name`
Modules

It seems modules must be compiled with your main program, unlike `included` files.

```bash
> f90 module.f90 program.f90 -o program
> f90 module.f90 -c
> f90 program.f90 module.o -o program
```

See example `RelativeHumidityUse.f90` and `atmosLib.mod`
Makefiles

Ok, I’m getting tired of typing out these long compiling commands.

Makefiles reduce typing, reduce risk of missing components, and help keep your source code and executables organized.

‘make’ is a unix command, so PC users will have to talk to their computer people about if it will work.

To use make, we create a makefile that describes how to compile our program and then just type

> make
> make -f makeFileName

and all of the required pieces of code get compiled
Makefiles

- Makefiles contain rules with the following shape:
  - `target ... : prerequisites ...
    command
    ...

- The commands *must be indented with a tab*

- Often, the makefile contains rules for the creation of many different programs or different source compilations AND rules for cleaning up.

- Check out the example

- Simplest if we keep one program per makefile, but can be extended to several (watch out for your ‘cleans!’)
Writing to Files

We learned the redirect output function, but there are better ways to write out data.

In order to write data to a file from a Fortran program, you need three lines of code:

```
OPEN (UNIT=num, FILE='name', FORM='...', status='...')
WRITE (UNIT=num,FMT=...) ...
CLOSE (UNIT=num)
```

These three statements tell the computer to open a file (load its structure into memory), write data to the file with a given format, and then close the file (move it from fast memory back to the harddrive).
Writing to Files

- The unit number allows for multiple files to be open at a time. Just make sure your WRITE command has the correct file identifier.

- The FORM keyword has two options 'FORMATTED' or 'UNFORMATTED' (we’ll start with formatted output)

- The STATUS keyword has four options ‘UNKNOWN’ (default), ‘OLD’, ‘NEW’, or ‘REPLACE’

- The FMT keyword is more complicated. It can be replaced with a *, which we do in the first example...
* - default or list-directed formatting

f (floating point) for real numbers, use: \((fw.d)\)
- \(w\) = total number of characters per number (incl spaces, decimal pt and sign)
- \(d\) = number of places after the decimal pt

e (exponential) for E+/− type numbers, use: \((ew.d)\)
- \(w\) = total number of characters per number
- \(d\) = number of digits in the mantissa (always normalized)

a (alphanumeric) for character strings (letters)

i (integer) for integers, option use: \((iw.d)\)
- Where \(d\) will pad the outputted number with zeros

See example – ComplexAtmos.f90
Unformatted Output

Unformatted output is just the straight-up binary representation of your data.

Using unformatted files is faster, and easy if you know what you put in there.

Virtually impossible for anybody else to use, though.

machine dependent, types are not obvious, can’t open the files and see what’s in them, can’t open the files in any software that can’t read binary (such as Excel), etc.

See example ReadUnformatted.f90
Reading data into Fortran is very similar to writing it out. We need three lines of code again:

```fortran
OPEN (UNIT=num, FILE='name', FORM='...', status='...')
READ (UNIT=num,FMT=...) ...
CLOSE (UNIT=num)
```

These three statements tell the computer to open a file (load its structure into memory), read data into your program's data structures with a given format, and then close the file (move it from fast memory back to the harddrive).
Reading in Files

- See WriteExample.f90 and ReadExample.f90

- Often if you get data from a large dataset, it will come with example code for how to read it in.

- Be careful to make sure your program's data structures are correct, Fortran won't double check this for you!
Closer Look at Strings

- We haven’t spent much time talking about the Character type yet.

- Just like other data types, letters are just numbers.

- Most characters are single 8 bit blocks of memory, so the typical basic character set supported by almost every computer now is called the ASCII character set, with 256 different characters.

Closer Look at Strings

In Fortran, the statement

```
Character :: CharName
```

Sets aside 8 bits for a single character to be stored inside. So, if we want more than one letter, we have to use an array of Characters. Fortran supports this datatype and calls it a “string”. You don’t need to use the Dimension attribute any more.

```
Character (50) :: StringName
```

Gives us a string of length 50.
Closer Look at Strings

- When you declare a string of a certain size, you can use fewer characters, but not more.

  - Character (50) :: New = ‘My new pretty string!’
  - New = ‘Another nice, new string. :)’

- We can extract subsets of strings, just like arrays.

  - New(9:12) = ‘food’

- We can put two strings together (called concatenation).

  - New = StringA//StringB
Useful String Manipulation Tools

Fortran offers several useful built-in string manipulation functions:

- LEN gives the length of a string (including unused characters).
- TRIM, shortens the string to include only the used characters.
- INDEX will give the starting position of a string within another string.
- SCAN searches for the number of occurrences of a char in a string.
- aChar(i) returns the char of an ascii value, iChar(c) returns the ascii value of a char.
A note on Multi-Dimensional Arrays

So far, we’ve seen one dimensional (lists or vectors) arrays, but you can have as many dimensions as you want.

```plaintext
integer, dimension(10,10,10,10) :: multiArray
real, dimension(lon,lat,height,time) :: chemData
character (50), dimension(10) :: labelArray

labelArray(1) = 'Pressure'

chemData(180,45,0,100) = 45.2
```
Multi-Dimensional Arrays

Array element ordering – indices vary slower as the dimension gets larger \{(1,1)(2,1)(3,1)(1,2)(2,2)(3,2)(1,3)(2,3)(3,3)\}

Unlike C(++,#) or Java, in Fortran, arrays are stored in Column-major order.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
Recursion
The Sierpinski Triangle

- Is a classic example of a fractal: Self-similarity at every scale
- We see a lot of fractals in the atmospheric sciences (clouds, turbulence, ocean mixing, snowflakes, strange attractors, etc).
- The Sierpinski triangle would be pretty hard to make with a DO Loop. But it’s easy enough with Recursion.
- Recursion happens when you call a function from inside that function. We can do that in Fortran!
Recursive Subroutines

In Fortran, Recursion is kind of a pain in the butt

Must declare a subroutine to be RECURSIVE so that it can see its own definition

Must be careful about pass-by-reference (when is your subroutine updating the variables, and what is it updating?)

Recursion is slower than iteration and requires more memory.

Examples: Fact.f90, Fibonacci.f90
HyperText Markup Language - not as much a programming language as a series of tags that describe formatting to a web browser.

Tags are usually of the form `<tag>text</tag>`

It is common to use style sheets to do formatting on websites now

There are tons of HTML help sites on the web, and style sheet help
<html>
  <head>
    <title>My Website</title>
  </head>
  <body>
    <h1>Some text</h1>
    <br{text <b>text</b> <i>text</i>
    <br><a href="page.html">link text</a>
    <br><img src="image.jpg">
  </body>
</html>
Covered Today...

- All about networking
- Using code from other files
- Makefiles
- Writing to and Reading from files
- User defined data types
- Strings!!
- Recursion
- a little HTML