C/C++ for Fortran Programmers

- If it's Turing Complete, you can do what you want ... eventually.

- There are no magic bullets:
  - But some problems are screws and some are nails. A well-stocked toolbox makes life easier.
  - You can write bad programs in any language
  - Real Programmers can write Fortran in any language

http://polar.ncep.noaa.gov/mmab/papers/tn186/
http://polar.ncep.noaa.gov/mmab/faqs/demos.html
/nwprod/lib/sorc/omblib/

- Fortran 66, 77, 90,
  - Watfor, Ratfor, Watfiv,
  - Vendor versions
- C, C++, Objective C
- Basic
- Pascal

- Java
- Logo
- VAX assembly, 68030 assembly

- Perl, javascript, IDL, Matlab, ...
- Lisp, Forth, Ada, Cobol, ...

- Python, ...

Fortran 'vs' C/C++

- High Optimization
- Beat on bunches of known numbers (numerical model)
- Deal with mathematical libraries

- High Flexibility
- Figure out what numbers to work with (satellite data flow)
- Deal with operating system, X, ...

C vs. C++

C ~ F90 (procedural, limited type structuring, ...)

C++ ~ F2003 (Object-oriented ...)

C: singular purpose, few or no structures, little expected descent
C++: general purpose, structures, operations, descendants

- Object Orientation:
  - What kinds of things are you working on?
  - What kinds of things do you do to them?
  - What kinds of things can they do?
- Inheritance -> Build up entities
- Overloading -> select function based on arguments
- Templating
- Encapsulation
- Virtual Classes

- Pass by Reference
- Pass by Value

```
tempate <class T>
class math_demo {
  private:
        ...
  protected:
    ...
  public:
    T value;
    math_demo();   -- Constructor
    T add(T );
    ...
};
```

```
int main(void) {
  math_demo x, y;
  x.value = 5;
  y.value = 2;
  x.add(y.value);
  printf("x now = %d\n",x.value);
  return 0;
}
```

Trivial here, but try it with avbuoy (buoy.h) instead

Class Inheritance

- grid2_base<T>
  - grid2<T>
    - metricgrid<T>
      - psgrid<T>
      - nam<T>
      - gaussian<T>
      - llgrid<T>
        - global_half<T>
        - global_12th<T>

T62L18 – 3d grid
T384L64
GFS3d
NAM3d

- Grid2_base:
  - 2d array of 'things'
  - read, write, subset, equate, ...
- grid2:
  - do math
- metric: (virtual)
  - points have a geophysical location
- llgrid: (ex)
  - points are arrayed on a particular type of projection
- global_12th: (ex)
  - points are on a 1/12th degree lat-long grid (in NCEP convention)
- gl_lambert_1km
  - Great lakes domain, 1 km lambert projection

Metric (virtual class)
latpt locate(ijpt) = 0;
fijpt locate(latpt) = 0;

*Base class demands operations of its descendants.
*Person implementing those operations is expert on how they work for *their* case
*User need know nothing beyond the above (vs. shapes of earth, ...)

But, also could add:
fijpt locate(latpt, r_earth, eccentricity);
fijpt locate(latpt, WGS84);

Leading to:

```
void gridup_satellite(metricgrid &x, avhrr &y, modis &z) {
...
ijlocation = x.locate(y.latpt);
x[ijlocation] = y.value;
...
x[z.latpt] += z.value;
}
```

Note:
& is pass by reference (Fortran-style)
–> C++ permits exact(ish) duplication of your Fortran expectations

- Parameters
- Points
- Mvector
- Grib

- Buoy
- Color
- Genes
- Grid_base
- ssmiclass

- grid_math

- Grid3
- Metricgrid

- ncepgrids
  - psgrid
  - llgrid
  - Cofs
  - Eta
  - Gaussian
  - lambert
  - resops

- Date
- Location
- ...

Note: buoy (c.f.) already has date-related info, but isn't using a date class. We can change this transparently to user (if we've written the class correctly in the first place)