DART Tutorial Section 11: Creating DART Executables
DART requires use of F90 *use module_mod, only*:
No other mechanism for use of ‘external’ routine.

Program viewed as Directed Acyclic Graph (tree with shared leaves)

No cycles!

Partial tree for program filter is shown.

Vector represents use of target module.
Generating a Makefile with \texttt{mkmf}

DART requires use of F90 \textit{use module\_mod, only}:

Use \texttt{mkmf} Perl script to make a Makefile

\texttt{mkmf} requires a list of files to search for the main program and modules that are used.

This is called a \texttt{path\_names} file.

See \texttt{path\_names\_filter} in \texttt{models/lorenz\_63/work}.
DART requires use of F90 *use module_mod, only*:

- **mkmf** searches files in `path_names` for one that contains *program filter*.
  - Finds first *use only* in `filter`.
  - Searches `path_names` files for this module recursively.
  - Builds a dependency graph like one at left.
Generating a Makefile with mkmf

DART requires use of F90 *use module_mod, only*:

From dependency graph, *mkmf* generates a standard *Makefile*.

Also creates a default namelist file, *input.nml.filter_default*.

Examine to see namelists required, default values.

Enter *make* to create filter executable.

DART Tutorial Section 11: Slide 5
Each DART program has \textit{mkmf} and \textit{path\_names} files. Can see a selection of these in \textit{models/lorenz\_63/work}.

Let’s look at \textit{mkmf\_perfect\_model\_obs} first.

File \textit{mkmf\_perfect\_model\_obs} starts with scripting that controls whether code is built with mpi for parallel runs.

Key line starts with “../../../build” and contains

(default values in parentheses):
1. Relative location of mkmf program (../../../build\_templates/mkmf)
2. Name of executable program to create (-p perfect\_model\_obs)
3. Compiler options file (-t ../../../build\_templates/mkmf.template)
4. Relative base location for file search (-a "../../../#"),
5. Name of the path\_names file (path\_names\_perfect\_model\_obs).
A variety of `mkmf` templates for different machines and compilers are available in the directory `build_templates` (You can also see the Perl script `mkmf` there).

`mkmf` templates specify:
- What compiler to use,
- Where to find the netCDF libraries,
- Compile and link command line options.

Templates usually include comments about useful compiler and linker options for both debugging and production.

To change your template:
Copy the appropriate `mkmf.template.xxx` file to `mkmf.template` and make any needed changes.
Two versions of the MPI module are available; one which really calls MPI and one with dummy routines.

The small models default to building a serial program. The large models default to building an MPI parallel version.

To run `mkmf` for filter, execute shell script `mkmf_filter`:

```
./mkmf_filter    OR    csh mkmf_filter
```
Can swap modules with same names and public interfaces. Changing the following paths in `path_names_filter`:

- `models/lorenz_63/model_mod.f90` switches to `models/cam-fv/model_mod.f90`
- `assimilation_code/location/oned/location_mod.f90` switches to `assimilation_code/location/threed_sphere/location_mod.f90`

Modules with multiple implementations have second directory level (see paths above).

Compare path_names files in `models/lorenz_63` with `models/cam-fv`
Exercise: Compiling lorenz_63 filter program

1. Go to `models/lorenz_63/work`

2. Remove all files with .o and .mod extensions.

3. Generate a `Makefile` and `input.nml.filter_default` by: 
   ```csh```
   mkmf_filter
   ```

4. Generate program filter:
   Enter `make`
01. Filtering For a One Variable System
02. The DART Directory Tree
03. DART Runtime Control and Documentation
04. How should observations of a state variable impact an unobserved state variable? 
   Multivariate assimilation.
05. Comprehensive Filtering Theory: Non-Identity Observations and the Joint Phase Space
06. Other Updates for An Observed Variable
07. Some Additional Low-Order Models
08. Dealing with Sampling Error
09. More on Dealing with Error; Inflation
10. Regression and Nonlinear Effects
11. Creating DART Executables
12. Adaptive Inflation
13. Hierarchical Group Filters and Localization
14. Quality Control
15. DART Experiments: Control and Design
16. Diagnostic Output
17. Creating Observation Sequences
18. Lost in Phase Space: The Challenge of Not Knowing the Truth
19. DART-Compliant Models and Making Models Compliant
20. Model Parameter Estimation
21. Observation Types and Observing System Design
22. Parallel Algorithm Implementation
23. Location module design (not available)
24. Fixed lag smoother (not available)
25. A simple 1D advection model: Tracer Data Assimilation