Make and Makefiles

Makefile Disclaimer

This course will give a brief overview of how to use make with Fortran

Will cover the basics only!

Then look at how modules complicate the use of make

What is Make?

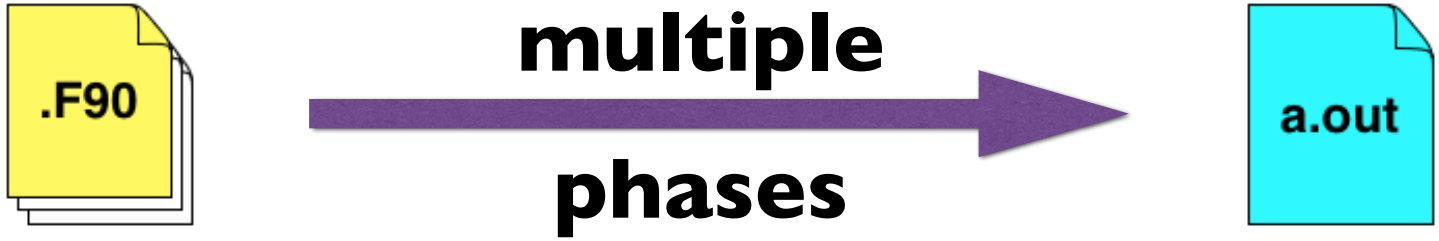
Make is a tool which controls the generation of executables from a program's source files

It gets its knowledge of how to build your program from a file called the makefile

The compilation procedure is much faster

- The compilation is done with a single command
- Only files that have been modified are recompiled
- Allows managing large programs with lots of dependencies

The Build Process



source code

executable

1. preprocessing phase
2. compiler phase
3. link phase

Fortran Preprocessing

Preprocessing serves multiple purposes:

- Macros for code simplification
- Inclusion of additional source files
- Conditional compilation

Present-day Fortran compilers will automagically handle this step for you. Just use .F90 or .F as the file extension.

Macros example: cpreproc.F90

Compilation Phase

source code

object files

Compilation consists of translating the individual plain text source files into machine code (or "object code")

Link Phase

object files

executable

Creates the executable file!

Makefile Basics (1)

A rule in the makefile tells Make how to execute a series of commands in order to build a target file from source files

It also specifies a list of prerequisites of the target file

Here is what a simple rule looks like:

```
target:  prerequisites … (also called dependencies)
    <tab> recipe
```

The <tab> is absolutely necessary!

Makefile Basics (2)

A target is usually the name of a file that is generated by the program. Typically it is the executable file we want to create, but it can also be object files or an action to carry out (e.g., clean).

A prerequisite is a file that is used as input to create the target. A target often depends on several files.

A recipe is an action that make carries out. A recipe may have more than one command either on the same line or each on it's own (with a TAB!)

Makefile Basics (3)

Make uses timestamps to locate the files that have been modified since the last time make was executed

By default when you type make it looks for the file makefile or Makefile. You can designate a specific name with make -f <makefilename>

Can also use macros to give names to variables within the makefile. NOTE these are case-sensitive!

If no specific target is given in the make command then Make starts with the first target listed in the makefile

Let's start with a very simple example (abc program).

Makefile Basics (4)

Comments are delimited by the # symbol

A backslash \ can be used as a continuation character

Common extra tidbit: Create a "phony target" called clean which can be run to do a fresh recompile of all source code

Makefile Automatic Variables

These can **only** be used in the recipe. They cannot be used in the target list of a rule

$< The name of the first prerequisite

$^ The names of the all prerequisites

$@ The file name of the target of the rule

And there are even more available

Compiling Modules

When modules are compiled both a .o and .mod file are created

A .mod file is like a compiled header. This is what the compiler searches for when it sees a USE statement

The dependencies can start to get cumbersome and complicated when many modules are USED and inherited

Make has no method for determining these for you.

Let's take a look at the other examples

Helpful Tools

mkDepends - generate a list of dependencies

mkSrcfiles - generate a list of all source files

Versions of these perl scripts are used in atmospheric models like SAM and CAM

mkdep - requires both GNU make and Python

fmfmk.pl - generate a makefile

foraytool - made especially for compiling large Fortran codes

Helpful Links

www.gnu.org - main GNU website

www.gnu.org/software/make - Make info

gcc.gnu.org/onlinedocs/gfortran - gfortran info

www.scons.org - Scons = software construction

www.cmake.org - cross-platform open-source build system