

Make and Makefiles

GNU Make

<http://www.gnu.org/software/make>

Dunfield Make

<http://www.dunfield.com>

Make Overview

- The make utility automatically determines which pieces of a program need to be recompiled, and issues commands to recompile them
- Make requires a makefile that describes the relationships between files in your program and provides commands for updating each file
- Once a suitable makefile exists, you only need to type 'make' or 'make target' on the command line in order to compile and link all files required for your program
- For more information, refer to the GNU make overview document (by Richard M. Stallman and Roland McGrath) available on the course web site. Parts of this presentation contain material from their GNU make overview document.
- Two make utilities are available for use in ECEN 5613. The free GNU make utility has many more features, is more widely used, and is more standardized. The Dunfield make utility is included with the licensed MICRO-C compiler tools, but is much less capable than GNU make.

Makefile Structure

- A simple makefile consists of "rules" with the following structure:

```
target ... : prerequisites ...  
          command  
...
```
- A **target** is usually the name of a file that is generated by a program; examples of targets are executable, object, or hex files. It can also be the name of an action to carry out, such as 'clean'
- A **prerequisite** is a file that is used as input to create the target. A target often depends on several files.
- A **command** is an action that make carries out. A rule may have more than one command, each on its own line. **Note: for GNU make you need to put a tab character at the beginning of every command line!**
- Usually a command is in a rule with prerequisites and serves to create a target file if any of the prerequisites change. However, the rule that specifies commands for the target need not have prerequisites. For example, the rule containing the delete command associated with the target 'clean' does not have prerequisites.
- A **rule**, then, explains how and when to remake certain files which are the targets of the particular rule. make carries out the commands on the prerequisites to create or update the target. A rule can also explain how and when to carry out an action.

(C) Copyright 2004 Linden H. McClure, Ph.D.

3

Using Make

- Dunfield make is a DOS utility; therefore filenames are subject to DOS 8.3 naming limitations
- Command line lengths are limited to 80 characters. For instance, when using slink.exe, you must make sure that the total expanded command line does not exceed 80 characters. You may need to use short file names in order to fit within the 80 character limit.
- Differences between GNU make and Dunfield make
 - GNU make processes only the specified/relevant targets in the makefile; if no target is specified, it processes the first target found in the makefile. Dunfield make processes all targets in the makefile sequentially each time you execute make.
 - GNU make requires a tab character at the beginning of each command line. Dunfield make uses spaces.
 - GNU make and Dunfield make use different syntax and symbols for macros/automatic variables
 - In the following examples, you would execute make as follows:
 - make (for Dunfield make)
 - make prog.hex (for GNU make)

(C) Copyright 2004 Linden H. McClure, Ph.D.

4

GNU Make Simple Example

```
#####
# GNU makefile demonstrating combination of C and assembly source files
# File Name: gnumakefile
# Only the targets which apply will be processed by GNU make.
# To create the file 'prog.hex' using GNU make, execute 'make prog.hex'.
# Note: GNU make requires a special format for the makefile:
#       A tab character must be at the beginning of every command line!
#####

# The following line declares a simply expanded variable
MCDIR := c:\mc

# Note: 'm=2' option defines the compact memory model for mcc51
# The following rule sends casm.c through the preprocessor and creates casm.tmp.
# casm.tmp is then compiled to create casm.asm.
# Note: Each MICRO-C option must use a separate '-' as shown below.
casm.asm : casm.c
    $(MCDIR)\mcp casm.c casm.tmp -l -c l=$(MCDIR)
    $(MCDIR)\mcc51 casm.tmp casm.asm -l -c -s m=2

# The following rule links casm.asm and myasm.asm and creates prog.asm.
# prog.asm is then assembled to create prog.hex (and prog.lst).
# Note: An external index file must be specified for slink using the 'i' option
prog.hex : casm.asm myasm.asm
    $(MCDIR)\slink casm.asm myasm.asm prog.asm l=$(MCDIR)\lib51 i=compact.lib
    $(MCDIR)\asm51 prog.asm -f -s

clean:
    del *.lst *.hex
```

(C) Copyright 2004 Linden H. McClure, Ph.D.

5

GNU Make Complex Example

```
#####
# GNU makefile demonstrating combination of C and assembly source files
# File Name: gnumakefile
# Only the targets which apply will be processed by GNU make.
# To create the file 'prog.hex' using GNU make, execute 'make prog.hex'.
# Note: GNU make requires a special format for the makefile:
#       A tab character must be at the beginning of every command line!
#####

# The following line declares a simply expanded variable
MCDIR := c:\mc

# Note: 'm=2' option defines the compact memory model for mcc51
# The following rule sends casm.c through the preprocessor and creates casm.tmp.
# casm.tmp is then compiled to create casm.asm.
# $@ is an automatic variable which contains the target name (casm.asm).
# The syntax $(@:.asm=.tmp) substitutes the suffix .tmp for the suffix .asm.
# $< is an auto variable containing the name of the first prerequisite (casm.c)
# Note: Each MICRO-C option must use a separate '-' as shown below.
casm.asm : casm.c
    $(MCDIR)\mcp $< $(@:.asm=.tmp) -l -c l=$(MCDIR)
    $(MCDIR)\mcc51 $(@:.asm=.tmp) $@ -l -c -s m=2

# The following rule links casm.asm and myasm.asm and creates prog.asm.
# prog.asm is then assembled to create prog.hex (and prog.lst).
# Note: An external index file must be specified for slink using the 'i' option
prog.hex : casm.asm myasm.asm
    $(MCDIR)\slink $^ $(@:.hex=.asm) l=$(MCDIR)\lib51 i=compact.lib
    $(MCDIR)\asm51 $(@:.hex=.asm) -f -s

clean:
    del *.lst *.hex
```

(C) Copyright 2004 Linden H. McClure, Ph.D.

6

Dunfield Make Simple Example

```
#####
# DDS makefile demonstrating combination of C and assembly source files
# File Name: makefile
# All targets in the makefile are processed sequentially by DDS make.
# To create the file 'prog.hex' using DDS make, just execute 'make'.
#####

# The following line declares a macro.
MCDIR=c:\\mc

# The following rule sends casm.c through the preprocessor and creates casm.tmp.
# casm.tmp is then compiled to create casm.asm.
# Note: 'm=2' option defines the compact memory model for mcc51
# Note: Each MICRO-C option must use a separate '-' as shown below.
casm.asm : casm.c
    $MCDIR\\mcp casm.c casm.tmp -l -c l=$MCDIR
    $MCDIR\\mcc51 casm.tmp casm.asm -l -c -s m=2

# The following rule links casm.asm and myasm.asm and creates prog.asm.
# prog.asm is then assembled to create prog.hex (and prog.lst).
# Note: An external index file must be specified for slink using the 'i' option
prog.hex : casm.asm myasm.asm
    $MCDIR\\slink casm.asm myasm.asm prog.asm l=$MCDIR\\lib51 i=compact.lib
    $MCDIR\\asm51 prog.asm -f -s
```

(C) Copyright 2004 Linden H. McClure, Ph.D.

7

Dunfield Make Complex Example

```
#####
# DDS makefile demonstrating combination of C and assembly source files
# File Name: makefile
# All targets in the makefile are processed sequentially by DDS make.
# To create the file 'prog.hex' using DDS make, just execute 'make'.
#####

# The following line declares a macro.
MCDIR=c:\\mc

# The following rule sends casm.c through the preprocessor and creates casm.tmp.
# casm.tmp is then compiled to create casm.asm.
# $@ is a macro symbol containing the name only of the target (casm).
# $. contains the full name of the each file in the dependency list (casm.c)
# Note: 'm=2' option defines the compact memory model for mcc51
# Note: Each MICRO-C option must use a separate '-' as shown below.
casm.asm : casm.c
    $MCDIR\\mcp $. $@.tmp -l -c l=$MCDIR
    $MCDIR\\mcc51 $@.tmp $@.asm -l -c -s m=2

# The following rule links casm.asm and myasm.asm and creates prog.asm.
# prog.asm is then assembled to create prog.hex (and prog.lst).
# Note: An external index file must be specified for slink using the 'i' option
prog.hex : casm.asm myasm.asm
    $MCDIR\\slink $. $@.asm l=$MCDIR\\lib51 i=compact.lib
    $MCDIR\\asm51 $@ -f -s
```

(C) Copyright 2004 Linden H. McClure, Ph.D.

8