make & project0

EECS 281: Data Structures & Algorithms

GNU make

Description: The make utility automatically
determines which pieces of a large
program need to be recompiled, and
issues commands to recompile them.

Man page:

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

Makefile

- GNU make can use any text file but defaults to Makefile (or makefile)
- A collection of definitions and build rules

Choosing a Specific g++ Version

```
 # Selects   g++ version   4.8.3,    allowing   c++11
PATH :=  /usr/um/gcc-4.8.3/bin:$(PATH)
LD_LIBRARY_PATH  := /usr/um/gcc-4.8.3/lib64
LD_RUN_PATH :=  /usr/um/gcc-4.8.3/lib64
# If  you  want  a  different  version, change  all
# lines  consistently.
```

Setting Variables

```
# TODO
# Change 'executable' to match the  command  name given      in  the
# project  spec.
EXECUTABLE  = executable

# designate which  compiler to  use
CXX             = g++
```

Setting Variables (cont.)

```
# list of  test  drivers (with  main()) for development
TESTSOURCES  = $(wildcard test*.cpp)
# names  of test  executables
TESTS  = $(TESTSOURCES:%.cpp=%)

# list  of  sources  used  in  project
SOURCES      = $(wildcard  *.cpp)
SOURCES      := $(filter-out  $(TESTSOURCES),  $(SOURCES))
# list  of  objects  used  in  project
OBJECTS      = $(SOURCES:%.cpp=%.o)
```

Setting Variables (cont.)

```
# TODO
# If main() is in a file named project*.cpp, use the following
line
PROJECTFILE = $(wildcard project*.cpp)
# TODO
# If main()    is in another file delete the  line  above, edit  and
# uncomment below
#PROJECTFILE  = mymainfile.cpp
```

Setting Variables (cont.)

```
#  name of  the  tar  ball  created for  submission
PARTIAL_SUBMITFILE  =  partialsubmit.tar.gz
FULL_SUBMITFILE  =  fullsubmit.tar.gz


#Default  Flags
CXXFLAGS  =  -std=c++11  -Wall  -Werror  -Wextra  -pedantic
```

GNU make Build Rules

- General format

  $<$target(s)$>$: $<$prerequisite(s)$>$

  <TAB>$<$recipe$>$

- Attempts to build $<$target(s)$>$ file(s)   from the $<$prerequisite(s)$>$ file(s), using   the command(s) in $<$recipe$>$

- USE A TAB CHARACTER!!

Build Rule Variations

- Filename wildcards: %.o, %.cpp, etc.
- Default rules
  - Implicit
  - Explicit
  - make -R -r
- Missing recipes

Basic Rules

```
# make release - will compile "all" with  $(CXXFLAGS)  and  the
# -O3  flag  also  defines NDEBUG  so  that  asserts will  not  check
release: CXXFLAGS  += -O3  -DNDEBUG
release: all


# make debug  - will compile  "all"  with  $(CXXFLAGS)  and  the
# -g  flag  also  defines  DEBUG  so  that  "#ifdef DEBUG"  works
debug:  CXXFLAGS  += -g3  -DDEBUG
debug:   clean  all


# make profile - will  compile  "all" with  $(CXXFLAGS)  and  the
# -pg  flag
profile: CXXFLAGS  += -pg
profile:  clean all
```

Rules Using Variables

```
#  highest target;  sews  together  all objects into  executable
all: $(EXECUTABLE)

$(EXECUTABLE):  $(OBJECTS)
ifeq ($(EXECUTABLE),  executable)
    @echo Edit  EXECUTABLE  variable, at  first     \"TODO\" in
    Makefile.
    @echo Using  default  a.out.
    $(CXX) $(CXXFLAGS)  $(OBJECTS)
else
    $(CXX) $(CXXFLAGS) $(OBJECTS)   -o  $(EXECUTABLE)
endif
```

Advanced make

```
# Automatically generate build rules for any test*.cpp files
define make_tests
    SRCS= $$(filter-out $$(PROJECTFILE), $$(SOURCES))
    OBJS = $$(SRCS:%.cpp=%.o)
    HDR = $$(wildcard *.h)
    $(1): CXXFLAGS += -g –DDEBUG
    $(1): $$(OBJS) $$(HDRS) $(1).cpp
    ifeq ($$(PROJECTFILE),)
      @echo Edit PROJECTFILE variable to .cpp file with main\(\)
      @exit 1
    endif
    $$(CXX) $$(CXXFLAGS) $$(OBJS) $(1).cpp -o $(1)
endef
```

Advanced make (cont.)

$(foreach test, $(TESTS), $(eval $(call make_tests, $(test))))

- make_tests uses the name of file matching test*.cpp to return a string with the proper build rule
- eval executes the string, and a build rule is created

Example: given testPQ.cpp

make_tests generates this equivalent build rule (simplified)

```
testPQ:    CXXFLAGS  += -pg  -DDEBUG
testPQ:    <all .o  and  .h files  minus  main  project .o  file>
      g++ <flags> <all  .o  files>  testPQ.cpp -o  testPQ
```

"make-ing" Testing Easy

- Just write test driver programs in test*.cpp
- make_tests creates individual build rules for each test*.cpp file

```
alltests: clean $(TESTS)
```

- This creates 'make alltests' to clean and build all tests created with make_tests at once!

An Explicit Default Build Rule

rule for creating objects
%.o: %.cpp
      $(CXX) $(CXXFLAGS) -c $*.cpp

- make knows how to go from .cpp to .o with an implicit default build rule

- 'make -R -r' disables all implicit build rules

Cleaning Up

```
#  make clean   - remove  .o files,      executables,     tarballs
clean:
        rm -f $(OBJECTS)   $(EXECUTABLE) $(TESTS)
$(PARTIAL_SUBMITFILE)  $(FULL_SUBMITFILE)
```

Building Submit Tarballs

\# make partialsubmit.tar.gz – remove old tarballs, run dos2unix,

\# create tarball omitting test cases and test drivers

PARTIAL_SUBMITFILES=$(filter-out $(TESTSOURCES), $(wildcard Makefile *.h *.cpp))

$(PARTIAL_SUBMITFILE): $(PARTIAL_SUBMITFILES)

 rm -f $(PARTIAL_SUBMITFILE) $(FULL_SUBMITFILE)

 dos2unix $(PARTIAL_SUBMITFILES)

 tar -vczf $(PARTIAL_SUBMITFILE) $(PARTIAL_SUBMITFILES)

 @echo !!! WARNING: No test cases included. Use 'make fullsubmit' to include test cases. !!!

Building Submit Tarballs (cont.)

make fullsubmit.tar.gz – remove old tarballs, run dos2unix,
create tarball including test cases omitting test drivers

FULL_SUBMITFILES=$(filter-out $(TESTSOURCES), $(wildcard Makefile *.h *.cpp test*.txt))

$(FULL_SUBMITFILE): $(FULL_SUBMITFILES)

      rm -f $(PARTIAL_SUBMITFILE) $(FULL_SUBMITFILE)

      dos2unix $(FULL_SUBMITFILES)

      tar -vczf $(FULL_SUBMITFILE) $(FULL_SUBMITFILES)

      @echo !!! Final submission prepared, test cases
included... READY FOR GRADING !!!

Building Submit Tarballs (cont.)

```
#  shortcut  for  make submit  tarballs
partialsubmit:  $(PARTIAL_SUBMITFILE)
fullsubmit:  $(FULL_SUBMITFILE)
```

- Now 'make partialsubmit' and 'make   fullsubmit' build tarballs, suitable for an autograder submission

Custom Prerequisites (Dependencies)

- Use these when new classes are created

- Build target is usually a .o file

- DON'T COMPILE *.h files!!!

- Previous rule for %.o can make a custom recipe unnecessary

Other make Directives

\# these targets do not create any files

.PHONY: all release debug profile clean alltests partialsubmit fullsubmit help

\# disable built-in rules

.SUFFIXES:

- .PHONY: this build rule doesn't create <targets> file(s)

- .SUFFIXES: disable implicit built-in rules

project0

- File available on CTools in
  Resources/Project/P0/project0.tgz
- A sample project solution
  - Makefile
  - project0.cpp
  - class.h, class.cpp
  - test_class.cpp
  - test_data.txt

project0 make Commands

- make release (or make all)
- make debug
- make profile
- make clean
- make test_class
- make alltests
- make partialsubmit
- make fullsubmit

$ make help
For more details!