CS 162
Intro to Computer Science II

Makefiles
Outline

• Terminology
• Makefile Contents
  – Rules
  – Targets
  – Dependencies
  – Variables
Basics

- Type “make” with no targets selected
- “make” searches for makefile or Makefile
- “make” executes the default action described in the makefile
Terminology

# This is a comment

target: dependencies

[tab] rule to build a program

MUST be a tab character, else it is ignored.
Sample Makefile

default: main.cpp functions.cpp functions.hpp
    g++ main.cpp functions.cpp -o program1

To run this makefile you just type: “make”
Dependencies

• Files required to build the program
• If any of the dependencies have timestamps newer than the target, it rebuilds the target
• In our previous slide, the target was called default which is not the name of the executable
• If you type make, it will always recompile
• To make it check timestamps, change the target to be the executable name ie. program1
Revised Makefile

program1: main.cpp functions.cpp functions.hpp
g++ main.cpp functions.cpp –o program1

• When you type “make” the first time, it compiles everything
• When you type “make” the second time it will say program1 is up to date
Variables

CXX = g++
SRCS = functions.cpp main.cpp
HEADERS = functions.hpp

program1: main.cpp functions.cpp functions.hpp
    g++ main.cpp functions.cpp —o program1

• You can declare variables in makefiles and use them
• (note these are only declared and haven’t actually used yet)
Variables

• Some standard C++ makefile variables:
  – CXX (C++ compiler)
  – CXXFLAGS (C++ compiler flags)
  – LDFLAGS (Linker flags)
  – OBJS (Object files)
  – SRCS (Source files)
  – HEADERS (Header files)
Variables

CXX = g++
SRCS = functions.cpp lecture1.cpp
HEADERS = functions.hpp

program1: ${SRCS} ${HEADERS}
  ${CXX} ${SRCS} –o program1

You must use the $ with braces, ${SRCS}, or parentheses $(SRCS)$ to indicate a variable
Variables

• Use += to concatenate to a variable eg.

  \[
  \text{CXXFLAGS} = -\text{std}=c++0x \\
  \text{CXXFLAGS} += -\text{wall} \\
  \text{CXXFLAGS} += -\text{pedantic}-\text{errors}
  \]

• This will result in

  \[
  \text{CXXFLAGS} = -\text{std}=c++0x -\text{wall} -\text{pedantic}-\text{errors}
  \]
Variables

CXX = g++
SRCS = functions.cpp main.cpp
HEADERS = functions.hpp
program1: ${SRCS} ${HEADERS}
    ${CXX} ${SRCS} –o program1

• If any file is changed, ALL of them will be recompiled
• This can take a long time for large programs
Variables

CXX = g++
HEADERS = functions.hpp
program1: functions.o main.o ${HEADERS}  
    ${CXX} functions.o main.o -o program1
functions.o: functions.cpp  
    ${CXX} -c functions.cpp
main.o: main.cpp  
    ${CXX} -c main.cpp

• By adding targets for the object files, only the source or header file that changed will be recompiled
• This requires a target for every object file!
Variables

• The solution is to add an implicit rule like
  
  \texttt{$\{OJBS\} : \{SRCS\}$}
  
  \texttt{$\{CXX\} -c \$(\@:.o=.cpp)$}
  
  that says the target is a .o file and the source is a .cpp file

• Don’t try to understand it.
• It’s magic! 😊
Touch

- if you want to update the timestamp on a file without doing anything to it, use “touch” on the UNIX command line:
  
  ```bash
  touch functions.cpp
  ```

- The Makefile will recompile targets that depend on functions.cpp
Targets

• Typing “make” executes the first target it finds
• If you want to execute a specific target eg. foo, you must type “make foo”
• Assuming that target is in your makefile
• To see this with the example Makefile, type in:
  touch functions.o
  make functions.o
# I removed the variables to fit this slide on the page

program1: ${OBJS} ${HEADERS}
     ${CXX} ${OBJS} –o program1
${OBJS}: ${SRCS}
     ${CXX} –c $(@:.o=:.cpp)
clean:
     rm –f * .o program1

• Removes object files and executable for a clean compile
• Type; “make clean”