

last time

building C code

compiling and linking

static and dynamic libraries

start on make

make rules

program: main.o extra.o

▶ clang -Wall -o program main.o extra.o

extra.o: extra.c extra.h

▶ clang -Wall -c extra.c

main.o: main.c main.h extra.h

▶ clang -Wall -c main.c

‘phony’ targets (1)

common to have Makefile targets that aren’t files

```
all: program1 program2 libfoo.a
```

“make all” effectively shorthand for “make program1
program2 libfoo.a”

no actual file called “all”

‘phony’ targets (2)

sometimes want targets that don’t actually build file

example: “make clean” to remove generated files

clean:

- ▶ `rm --force main.o extra.o`

but what if I create...

clean:

► `rm --force main.o extra.o`

`all: program1 program2 libfoo.a`

Q: if I make a file called “all” and then “make all” what happens?

Q: same with “clean” and “make clean”?

marking phony targets

clean:

► `rm --force main.o extra.o`

`all: program1 program2 libfoo.a`

.PHONY: all clean

special .PHONY rule says “ ‘all’ and ‘clean’ not real files”

(not required by POSIX, but in every make version I know)

conventional targets

common convention:

target name	purpose
(default), all	build everything
install	install to standard location
test	run tests
clean	remove generated files

redundancy (1)

program: main.o extra.o

► clang -Wall -o program main.o extra.o

extra.o: extra.c extra.h

► clang -Wall -o extra.o -c extra.c

main.o: main.c main.h extra.h

► clang -o main.o -c main.c

what if I want to run clang with `-fsanitize=address` instead of `-Wall`?

what if I want to change clang to gcc?

variables/macros (1)

CC = gcc

CFLAGS = -Wall -pedantic -std=c11 -fsanitize=address

LDFLAGS = -Wall -pedantic -fsanitize=address

LDLIBS = -lm

program: main.o extra.o

▶ \$(CC) \$(LDFLAGS) -o program main.o extra.o \$(LDLIBS)

extra.o: extra.c extra.h

▶ \$(CC) \$(CFLAGS) -o extra.o -c extra.c

main.o: main.c main.h extra.h

▶ \$(CC) \$(CFLAGS) -o main.o -c main.c

aside: conventional names

chose names CC, CFLAGS, LDFLAGS, etc.

not required, but conventional names (incomplete list follows)

CC	C compiler
CFLAGS	C compiler options
LDFLAGS	linking options
LIBS or LDLIBS	libraries

variables/macros (2)

```
CC = gcc
CFLAGS = -Wall
LDFLAGS = -Wall
LDLIBS = -lm
```

`$@`: target
`$<`: first dependency
`^`: all dependencies

```
program: main.o extra.o
```

```
▶ $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS)
```

```
extra.o: extra.c extra.h
```

```
▶ $(CC) $(CFLAGS) -o $@ -c $<
```

```
main.o: main.c main.h extra.h
```

```
▶ $(CC) $(CFLAGS) -o $@ -c $<
```

aside: `^` works on GNU make (usual on Linux), but not portable.

pattern rules

CC = gcc

CFLAGS = -Wall

LDFLAGS = -Wall

LDLIBS = -lm

program: main.o extra.o

▶ \$(CC) \$(LDFLAGS) -o \$@ \$^ \$(LDLIBS)

%.o: %.c

▶ \$(CC) \$(CFLAGS) -o \$@ -c \$<

extra.o: extra.c extra.h

main.o: main.c main.h extra.h

built-in rules

'make' has the 'make .o from .c' rule built-in already, so:

```
CC = gcc
```

```
CFLAGS = -Wall
```

```
LDFLAGS = -Wall
```

```
LDLIBS = -lm
```

```
program: main.o extra.o
```

```
▶      $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS)
```

```
extra.o: extra.c extra.h
```

```
main.o: main.c main.h extra.h
```

(don't actually need to write supplied rule!)

built-in rules

'make' has the 'make .o from .c' rule built-in already, so:

```
CC = gcc
```

```
CFLAGS = -Wall
```

```
LDFLAGS = -Wall
```

```
LDLIBS = -lm
```

```
program: main.o extra.o
```

```
▶ $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^ $(LDLIBS)
```

```
extra.o: extra.c extra.h
```

```
main.o: main.c main.h extra.h
```

(don't actually need to write supplied rule!)

note: built-in rules not allowed on the make lab

writing Makefiles?

error-prone to write all .h dependencies

- MM (and related) options to gcc or clang
 - outputs make rule
 - ways of having make run this + use output

Makefile generators

other programs that write Makefiles

other build systems

alternatives to writing Makefiles:

other make-ish build systems

ninja, scons, bazel, maven, xcodebuild, msbuild, ...

tools that generate inputs for make-ish build systems

cmake, autotools, qmake, ...