CSc 352

Make

Russell Lewis (sub for Benjamin Dicken)

Announcements

- PA 7 split into two
  - PA 7 due as planned, but only includes some of the functions
  - PA 8 due date TBD, but includes everything

- Ben back 4 Apr

- Exam 2: 6 Apr

Builds are Annoying!

- Problem: How to build reliably?

- By hand
  - Error prone
  - Have to teach new users
- Scripts
  - Non-standard output
  - Hard to update, confusing
  - Partial builds hard

Typical C Program

- Multiple Compilation
  - Build various .c files into .o files
  - Link together all .o files as a single, unified step

```
gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
gcc -Wall -Werror -std=c11 -c baz.c -o baz.o
gcc -lm foo.o bar.o baz.o -o my_program
```

Huge Projects

- My work @ IBM: 1800 C files, 3500 headers
- 200 programmers making changes to the central repository
- 30-60 minute build time

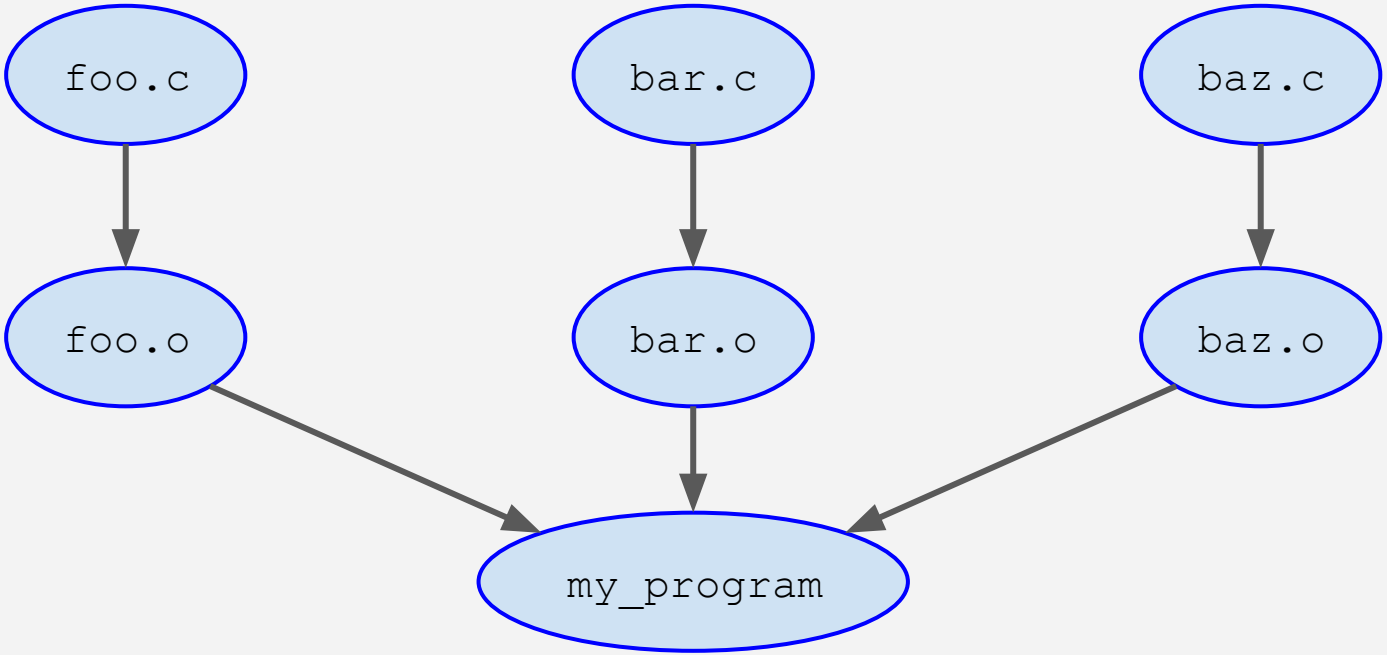
- Impractical for development!
  - Need to make small changes, compile, test
  - Can we rebuild only what has changed?

- Insight: dependencies as a tree!

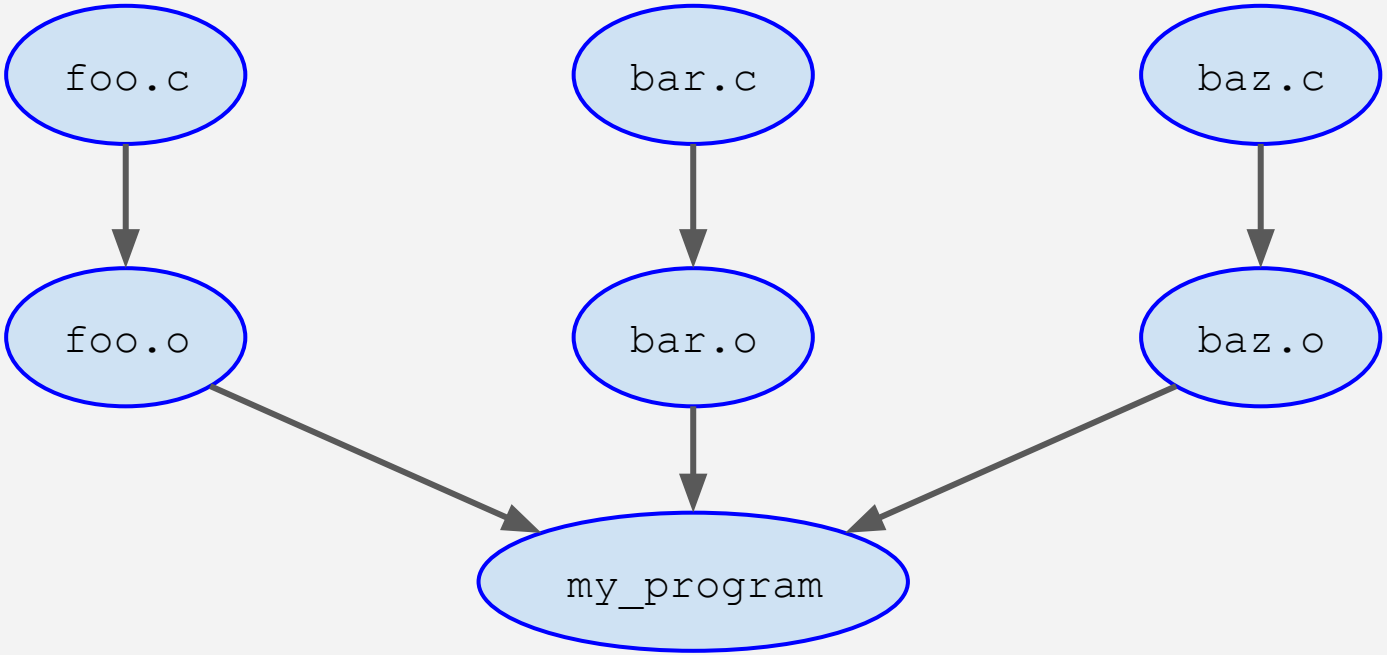
Dependency Trees

- Draw a tree that represents the **dependencies between files** in this build process
  - Child: input file
  - Parent: output file

```
gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
gcc -Wall -Werror -std=c11 -c baz.c -o baz.o
gcc -lm foo.o bar.o baz.o -o my_program
```

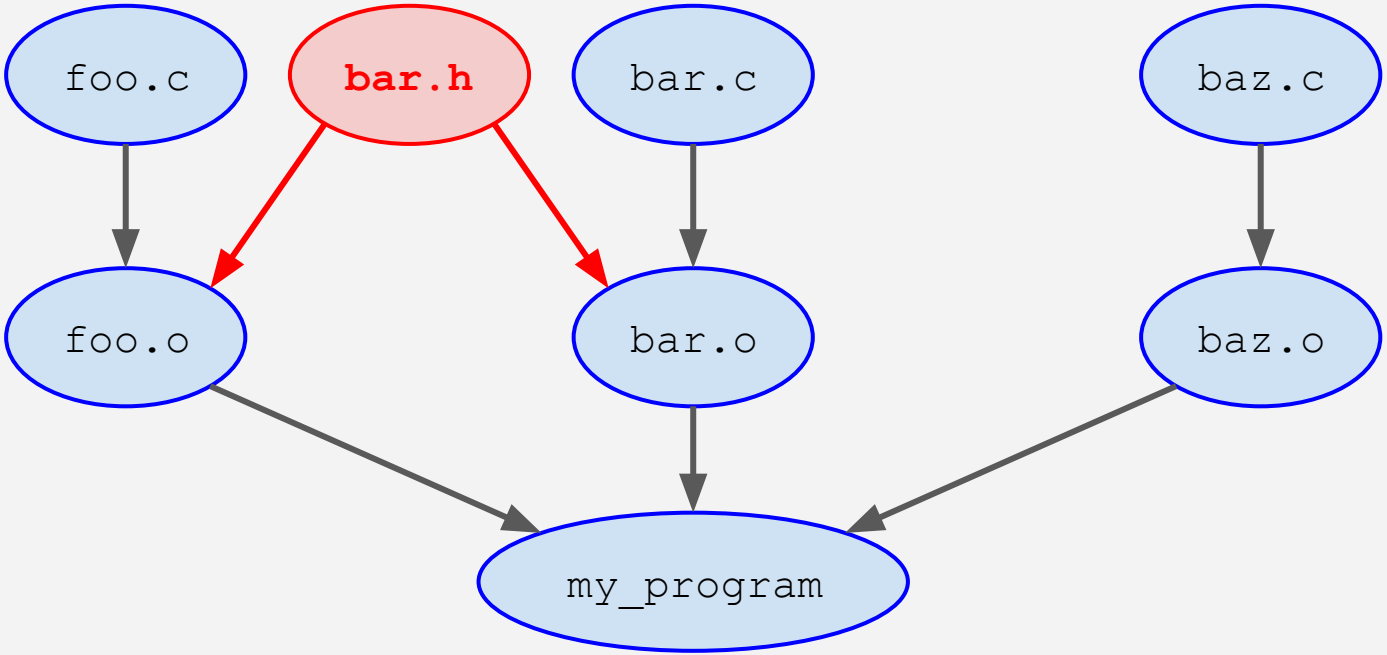



```
gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
gcc -Wall -Werror -std=c11 -c baz.c -o baz.o
gcc -lm foo.o bar.o baz.o -o my_program
```




```
gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
gcc -Wall -Werror -std=c11 -c baz.c -o baz.o
gcc -lm foo.o bar.o baz.o -o my_program
```

What about headers?



- Not actually a tree (why?)
- In truth, this is a DAG (directed acyclic graph)

Makefiles

- List a set of output files (or rules)
- Each has dependencies
- Most have *rules* to build them
  - Rules are shell commands

- Many, many advanced features

```
my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program

foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o

bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o

baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o
```

```
my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program

foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o

bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o

baz.o: baz.c
	gcc -Wall -Werror -std=c11
```

Outputs

Formally called “targets”

```
my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program

foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o

bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o

baz.o: baz.c
	gcc -Wall -Werror -std=c11 -
```

Inputs (dependencies)

Notice that header files are included!

```
my_program: foo.o bar.o baz.o
        gcc -lm foo.o bar.o baz.o -o my_program

foo.o: foo.c bar.h
        gcc -Wall -Werror -std=c11 -c foo.c -o foo.o

bar.o: bar.c bar.h
        gcc -Wall -Werror -std=c11 -c bar.c -o bar.o

baz.o: baz.c
        gcc -Wall -Werror -std=c11 -
```

Question:
Why not put **all** headers in the list for all .o files?

```
my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program

foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o

bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o

baz.o: baz.c
	gcc -Wall -Werror -st
```

WARNING:
Indentation before rules *must* be a tab, not spaces

`all, clean`, and Other Non-file Targets

- It’s common to have some rules which don’t actually produce the required file
  - If so, it is *always* out of date, so we always run the rules (if we need the target)
  - Useful for command-line use:
    - “make all” - often used to build the output tool(s)
    - “make clean” - removes all outputs
    - “make test” - run automated tests
    - “make install” - copy outputs to system directories

    etc.

A More Complete Makefile

- Fill out our example Makefile with more rules:
  - `all` builds my_program
  - `clean` removes all object files and the program

```
all: my_program

my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program
foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o

clean:
	-rm foo.o bar.o baz.o my_program 2>/dev/null
```

```
all: my_program

my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o
foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o

clean:
	-rm foo.o bar.o baz.o my_program 2>/dev/null
```

Question:
Why have an `all` rule that is separate from `my_program` ?

```
all: my_program

my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program
foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o

clean:
	-rm foo.o bar.o baz.o my_program 2>/dev/null
```

Question:
What does the hyphen in the `clean` rule do?

```
all: my_program

my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program
foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o

clean:
	-rm foo.o bar.o baz.o my_program 2>/dev/null
```

Question:
Why redirect `stderr`?

Variables

- Very useful to have various variables
  - Lists of object files
  - Common flags & commands

    etc.
- Setting a variable

```
OBJS=foo.o bar.o baz.o
```

- Use variables in dependency lists, target names, or rules:

```
my_program: $OBJS
        gcc -lm $OBJS -o my_program
```

- Use `$(NAME)` syntax if end-of-variable-name is ambiguous

Advanced Topics: Pattern Matching

- Pattern-matching rules allow you to define a common rule for a **type** of file - so long as the input and output share a base name

```
%.png: %.dot
        dot -Tpng -o $*.png $<
```

- `$*` - matches the base name
- `$<` - matches the (first) input file

Advanced Topics: Default Rules

- make provides a set of default pattern-matching rules for turning one type of file into another
  - No Makefile required!
  - Usually, good-enough for basic work

- Example: If `foo.c` exists, then
  - `make foo.o` will auto-compile it to an object file
  - `make foo` will auto-compile it to a program

Advanced Topics: Wildcards & Substitutions

- Wildcard matching allows you to find a list of files in a directory

```
C_FILES=$(wildcard *.c)
```

- You can do substitutions, based on patterns, to build new variables

```
OBJ_FILES=$(C_FILES:.c=.o)
```

Advanced Topics: Dependencies, Separated

- It's permissible to define some dependencies in one place in the file, and to define the rule elsewhere.

```
foo.o: foo.c bar.h
   ...many lines later...
foo.o:
    gcc -Wall -Werror -std=c11 foo.c -o foo.o
```

- (Also works with pattern-matching rules)

- Used in conjunction with make's `include` directive and gcc's `-MM` option, can be used for fully-automated dependency graphs

Optimized Makefile

- Bring up our old Makefile, and make it as cool and adaptable as you can! How many of the Advanced Features can you use?

```
all: my_program

my_program: foo.o bar.o baz.o
	gcc -lm foo.o bar.o baz.o -o my_program
foo.o: foo.c bar.h
	gcc -Wall -Werror -std=c11 -c foo.c -o foo.o
bar.o: bar.c bar.h
	gcc -Wall -Werror -std=c11 -c bar.c -o bar.o
baz.o: baz.c
	gcc -Wall -Werror -std=c11 -c baz.c -o baz.o

clean:
	-rm foo.o bar.o baz.o my_program 2>/dev/null
```

This is what we had before. Let's discuss how to make it better.

```
C_FLAGS=-Wall -Werror -std=c11

C_FILES=$(wildcard *.c)
OBJ_FILES=$(C_FILES:.c=.o)
%.o: %.c
    gcc $(CFLAGS) -c $< -o $*

all: my_program

my_program: foo.o bar.o baz.o
        gcc -lm $(OBJS) -o my_program

clean:
        -rm $(OBJS) my_program 2>/dev/null

foo.o bar.o: bar.h      # TODO: autogenerate me
```