Build systems and Make

Class 8

Overview

1. Announcements
2. Review
3. Q&A
4. Demo
5. Basic assignment

Announcements

- Git 2 assignments due Nov 8
- Editors assignments due Nov 15
- Make assignments due Nov 22
- GitLab rerun on at 12 AM December 21st
  - Get your assignments squared away by 11:59 PM December 20th
  - Rerun from a clean slate: no late penalties, but no regrades after the fact

Review

- Makefiles specify how to build files and how they depend on each other
- Makefiles are composed of **rules**

```
target: prerequisites
    recipe # <- actual tab character, not spaces!
```

- **Target**: file to be produced
- **Prerequisites**: list of files that the rule depends on
- **Recipe**: shell commands that build the target file

Review

```
file1.o: file1.cpp
    g++ -c -o file1.o file1.cpp

file2.o: file2.cpp
    g++ -c -o file2.o file2.cpp

file3.o: file3.cpp
    g++ -c -o file3.o file3.cpp

app: file1.o file2.o file3.o
    g++ -o app file1.o file2.o file3.o
```

Review

- Phony targets are targets that aren't files
  - Can represent ideas/concepts/commands e.g. `all`, `clean`, `test`
- Specify a target as phony by adding a `.PHONY` rule with the target as a prerequisite

Review

```
.PHONY: all
all: app

.PHONY: clean
clean:
    rm -f app

file1.o: file1.cpp
    g++ -c -o file1.o file1.cpp

file2.o: file2.cpp
    g++ -c -o file2.o file2.cpp

file3.o: file3.cpp
    g++ -c -o file3.o file3.cpp

app: file1.o file2.o file3.o
    g++ -o app file1.o file2.o file3.o
```

Review

- Two types of variables you can assign
  - Recursively expanded (via `=`)
  - Simply expanded (via `:=`)
- Automatic variables
  - `$@`, `$<`, `$^`, and more

Review

```
.PHONY: all
all: $(BIN)

.PHONY: clean
clean:
    rm -f $(BIN)

CXX = g++
BIN = app

file1.o: file1.cpp
    $(CXX) -c -o $@ $<

file2.o: file2.cpp
    $(CXX) -c -o $@ $<

file3.o: file3.cpp
    $(CXX) -c -o $@ $<

$(BIN): file1.o file2.o file3.o
    $(CXX) -o $@ $^
```

Review

- Make provides functions to help with text manipulation and other tasks
  - `$(wildcard <pattern>)`
  - `$(shell find . -name "*.c")`\`
  - \`$(dir $@)
- Make also provides pattern substitution and matching functionality
  - `$(<var>:<pattern>=<replacement>)` (substitution reference)
  - `$(SOURCES:%.c=%.o)`

```
%.o : %.c
    $(CC) -c -o $@ $<
```

```
OBJS := $(SRCS:src/%.c=obj/%.o) # substitution reference
$(OBJS): obj/%.o : src/%.c # static pattern rule
    $(CC) -c -o $@ $<
```

Review

```
.PHONY: $(BIN)
all: app

.PHONY: $(BIN)
clean:
    rm -f app

CXX = g++
BIN = app
SRCS = $(find . -name "*.cpp")
OBJS = $(SRCS:%.cpp=%.o)

$(OBJS): %.o: %.cpp
    $(CXX) -c -o $@ $<

$(BIN): $(OBJS)
    $(CXX) -o $@ $^
```

Q&A

Demo

Basic assignment