Build systems and Make

Class 7

Overview

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

Announcements

- Make assignments due April 2
- `eecs201-test calculate` to calculate grades

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

- Download the following file

```
https://www.eecs.umich.edu/courses/eecs201/wn2025
|- /files/examples/make2.tar.gz
```

- Lets put together a Makefile for this

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

Review

- Now let's make it do incremental building

```
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 file2.o file3.o

.PHONY: test
test:
    ./test 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)

.PHONY: test
test:
    ./test $(BIN)
```

Review

```
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

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

.PHONY: all
all: $(BIN)

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

.PHONY: test
test:
    ./test $(BIN)

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

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

Review

- Try splitting into directories
- Try adding flags

Review

```
CXX = g++
CXXFLAGS = -O3
BIN = app
SRCDIR = src
INCDIR = inc
OBJDIR = obj
SRCS = $(shell find $(SRCDIR) -name "*.cpp")
OBJS = $(SRCS:$(SRCDIR)/%.cpp=$(OBJDIR)/%.o)

.PHONY: all
all: $(BIN)

.PHONY: clean
clean:
    rm -rf $(BIN) $(OBJDIR)
```

Review

```
.PHONY: test
test:
    ./test $(BIN)

$(OBJS): $(OBJDIR)/%.o: $(SRCDIR)/%.cpp
    mkdir -p $(dir $@)
    $(CXX) $(CXXFLAGS) -I$(INCDIR) -c -o $@ $<

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

Q&A

Basic assignment