MAKEFILES

CS 23200

Outline

- Makefile basics
  - targets
  - out of date
  - dependencies
- Helpful Makefile tricks
  - Variables (i.e., macros)
  - Default compilation rules
  - make clean

make

- The `make` program helps you automate compiling and linking
- Simplifies compilation commands
- Will avoid compiling things that are already up-to-date

make Vocabulary

- target
  - A task that needs to be performed
  - Is often a file name to create
    - The target image.o would correspond to the task of compiling image.o
    - The target myProg would correspond to the task of compiling and linking myProg

make Vocabulary

- makefile
  - A text file
  - Specifies a set of targets and the commands for each target
  - Can be named anything, but convention is to name it "makefile" or "Makefile"
    - make looks for these files automatically

Makefile Example

```
# A very simple makefile example
# Lines beginning with # are comments
# Targets begin at the left margin, followed by :
# Shell command lines must begin with a TAB

imageList:
	gcc -g image.c list.c imageList.c -o imageList
	#can issue more commands for this target

image.o:
	gcc -g -c image.c -o image.o
```

Running make

```
# Shell command lines must begin with a TAB

imageList:
	gcc -g image.c list.c imageList.c -o imageList

image.o:
	gcc -g -c image.c -o image.o
```

- Run make from the shell
  - Without any arguments: makes the first target in the file

```
[shell prompt $]  make
```

Running make

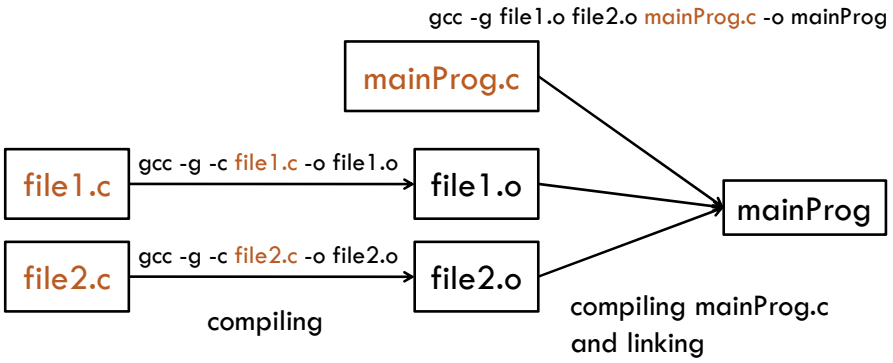
```
# Shell command lines must begin with a TAB

imageList:
	gcc -g image.c list.c imageList.c -o imageList

image.o:
	gcc -g -c image.c -o image.o
```

- Run make from the shell
  - With a target name: makes that target

```
[shell prompt $]  make image.o
[shell prompt $]  make imageList
```

Avoiding Recompilation

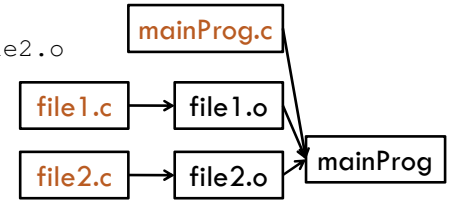
- mainProg.c changes: file1.o and file2.o are still "up to date"
- file1.c changes: file1.o is out of date, as is mainProg



Dependencies

- Need to tell make what targets to recompile when something changes
- file1.o depends upon file1.c, file1.h
- mainProg depends upon mainProg.c, file1.o, file2.o

```
# dependencies follow the :
mainProg: file1.o file2.o mainProg.c
      gcc -g file1.o file2.o mainProg.c -o mainProg
file1.o: file1.c file1.h
      gcc -g -c file1.c -o file1.o
file2.o: file2.c file2.h
      gcc -g -c file2.c -o file2.o
```



- When file2.c is out of date, make will start on the first target (mainProg):
  - Check that its dependencies are up to date
    - file1.o : check that its dependencies are up to date
      - file1.c : yes
      - file1.h : yes
    - file2.o : check that its dependencies are up to date
      - file2.c : NO --> will **rebuild the target file2.o**
    - mainProg.c : up to date
  - A dependency changed, so **rebuild the target mainProg**

```
# dependencies follow the :
mainProg: file1.o file2.o mainProg.c
      gcc -g file1.o file2.o mainProg.c -o mainProg
file1.o: file1.c file1.h
      gcc -g -c file1.c -o file1.o
file2.o: file2.c file2.h
      gcc -g -c file2.c -o file2.o
```

Outline

- Makefile basics
  - targets
  - out of date
  - dependencies
- **Helpful Makefile tricks**
  - Variables (i.e., macros)
  - Default compilation rules
  - make clean

A Typical Example

- Lots of .c files, each with their own header
- Compile each to an object file
- Compile and link all objects with a mainProg.c

- Want to avoid repeating basically the same thing for every object file:

```
file1.o: file1.c file1.h
	gcc -g -c file1.c -o file1.o
file2.o: file2.c file2.h
	gcc -g -c file2.c -o file2.o
file3.o: file3.c file3.h
	gcc -g -c file3.c -o file3.o
...
```

Example GNU Makefile

- `$(VARIABLE_NAME)` : replaced with string
- `$<` : the name of the first dependency
- `$@` : the target name
- `%` : a (GNU-specific) wildcard
- `$*` : the target name with suffix deleted

```
CC = gcc
CFLAGS = -g

OBJS = file1.o file2.o file3.o file4.o file5.o

mainProg: mainProg.c $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) $< -o $@
%.o : %.c %.h
	$(CC) $(CFLAGS) -c $< -o $@
```

Example GNU Makefile

- `clean` target typically removes binary files (object files and executables)
- `all` target typically makes every target
  - Often listed first

```
CC = gcc
CFLAGS = -g
OBJS = file1.o file2.o file3.o file4.o file5.o

all: mainProg
mainProg: mainProg.c $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) $< -o $@
%.o : %.c %.h
	$(CC) $(CFLAGS) -c $< -o $@
clean:
	rm *.o *~ mainProg
```