CS 50:
Software Design and Implementation

Make and Makefiles

Agenda

1. Makefiles

2. Compiling bagsimple with make

3. Activity

When programs become large it becomes difficult to correctly compile them

Even our bag module is starting to get complex to compile!

- `bagsimple.c` - a simple example of an application that uses the bag module
- `bag.h` - declarations that form the interface to the *bag* module
- `bag.c` - functions that define the implementation of the *bag* module.
- Since we also use the `readlinep` module, we must now compile the program with a command like:

```
$ mygcc –o bagsimple bag.c bagsimple.c readlinep.c
```

1. Remembering to include all needed files starts to become difficult
2. Plus if there aren't any changes to a file, no need to recompile it
   - This isn't a problem with the examples we've seen so far – they each take a few seconds to compile
   - Larger projects (say the Linux kernel) can take hours to compile

A Makefile solves these problems!

https://xkcd.com/303/

The `make` program reads a file called `Makefile` and runs commands in `Makefile`

By default, the `make` command looks for a file called `Makefile` (can change name of file with –f)

Makefile must follow specific syntax

Target is what to create

```
target: dependent files
	command 1
	command 2
```

Dependent files are those needed to create the target

Commands to run to create target

Must begin with a tab (spaces do not work!)

The make program reads a file called Makefile and runs commands in Makefile

By default the make command looks for a file called Makefile (can change name of file with –f)

Makefile must follow specific syntax

target: dependent files
        command 1
        command 2

File is called Makefile

Makefile

```
dumplings: veggies flour
	@echo "Making dumplings"

veggies:
	@echo "Buying vegetables"

flour:
	@echo "Buying flour"
```

To make dumplings we will need vegetables and flour

Gives what is needed to create veggies and flour

Dumplings target "depends" on veggies and flour

```
$ make dumplings
Buying vegetables
Buying flour
Making dumplings
$ make flour
Buying flour
$ make veggies
Buying vegetables
$ make
Buying vegetables
Buying flour
Making dumplings
```

make dumplings cause make to create veggies and then flour

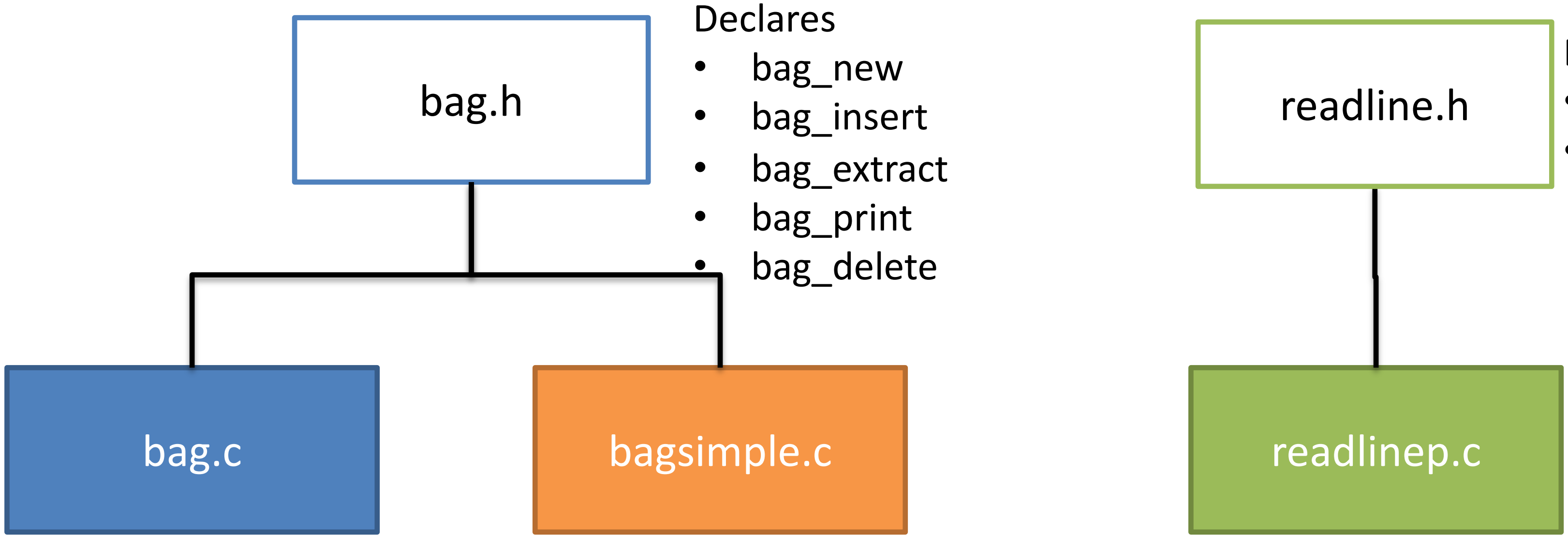
You can also execute veggies and flour targets on their own

Running make without a target runs the first target (dumplings here)

Agenda

1. Makefiles
2. Compiling bagsimple with make
3. Activity

Overview: read names from stdin and store in a bag, print and free bag when done



Declares
- bag_new
- bag_insert
- bag_extract
- bag_print
- bag_delete

Declares
- freadlinp
- readlinep

Includes bag.h
Implements
- bag_new
- bag_insert
- bag_extract
- bag_print
- bag_delete

Includes bag.h and readlinep.h
Uses implementation from bag.c
1. Creates new bag (bag_new)
2. Reads names of arbitrary length from stdin (readlinep)
   - Loop until control-D
   - Add each name (bag_insert)
3. Print all names in the bag (bag_print)
4. Delete the bag and free all dynamic memory (bag_delete)

Includes readlinep.h
Implements
- freadlinep
- Readlinep (freadlinep but from stdin)

```
$ mygcc -o bagsimple bag.c readlinep.c bagsimple.c
$ ./bagsimple
```

Compiling the bagsimple from last class becomes somewhat tedious

```
$ mygcc -o bagsimple bagsimple.c bag.c readlinep.c
```

Output executable as bagsimple

Alias we set up in bash_profile
alias mygcc='gcc -Wall -pedantic -std=c11 -ggdb'

Compile and link into bagsimple executable

- **bagsimple.c** (includes bag.h and readlinep.h)
- **bag.c** (includes bag.h)
- **readlinep.c** (includes readlinep.h)

- Starting to get complicated to type
- If one file changes, need to re-compile all files
- We can do better!
- We will use make and Makefiles from now on

A Makefile gives instructions on how to compile targets based on dependencies

Makebag1

Bagsimple target depends on object files from bag modules and readlinep

```
bagsimple: bag.o bagsimple.o readlinep.o
        gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

-c flag stops compilation after .o produced

```
bag.o: bag.c bag.h
        gcc -c bag.c
```

bag.o depends on bag.c and bag.h

If they change, recompile bag.c

```
bagsimple.o: bagsimple.c bag.h readlinep.h
        gcc -c bagsimple.c

readlinep.o: readlinep.c readlinep.h
        gcc -c readlinep.c
```

Once all the .o files are update to date, link them together into an executable called bagsimple

Bagsimple depends on bagsimple.c plus two headers, bag.h and readlinep.h (run `head -15 bagsimple.c`)

If those files change, recompile bagsimple.c to object file

Readlinep.o depends on readlinep.c and readlinep.h

If those files change, recompile readlinep.c to object file

A Makefile gives instructions on how to compile targets based on dependencies

Makebag1

```
bagsimple: bag.o bagsimple.o readlinep.o
	gcc -o bagsimple bag.o bagsimple.o readlinep.o

bag.o: bag.c bag.h
	gcc -c bag.c

bagsimple.o: bagsimple.c bag.h readlinep.h
	gcc -c bagsimple.c

readlinep.o: readlinep.c readlinep.h
	gcc -c readlinep.c
```

```
$ make -f Makebag1
gcc -c bag.c
gcc -c bagsimple.c
gcc -c readlinep.c
gcc -o bagsimple bag.o bagsimple.o readlinep.o
$ make -f Makebag1
make: 'bagsimple' is up to date.
```

-f flag tells Makefile to use Makebag1
If omitted, make looks for a file named Makefile
If do not specify target, make runs the first one

No need to recompile
Everything is up to date

Make knows .o files come from .c files!

Makebag1a

```
bagsimple: bag.o bagsimple.o readlinep.o
	gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

Make knows .o files come from .c files
If make can not find .o file, it will compile .c file with same name to make a .o file

Make knows .o files come from .c files!

Makebag1a

```
bagsimple: bag.o bagsimple.o readlinep.o
    gcc –o bagsimple bag.o bagsimple.o readlinep.o
```

Remove .o so make will recompile dependencies

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
```

Make knows .o files come from .c files!

Makebag1a

```
bagsimple: bag.o bagsimple.o readlinep.o
   gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

Remove .o so make will recompile dependencies

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
$ make -f Makebag1a
cc    -c -o bag.o bag.c
cc    -c -o bagsimple.o bagsimple.c
cc    -c -o readlinep.o readlinep.c
$ gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

Make compiles .c that matches .o

Make knows .o files come from .c files!

Makebag1a

```
bagsimple: bag.o bagsimple.o readlinep.o
   gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

Remove .o so make will recompile dependencies

Make compiles .c that matches .o

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
$ make -f Makebag1a
cc    -c -o bag.o bag.c
cc    -c -o bagsimple.o bagsimple.c
cc    -c -o readlinep.o readlinep.c
$ gcc -o bagsimple bag.o bagsimple.o readlinep.o
$ touch bag.c
$ make -f Makebag1a
cc    -c -o bag.o bag.c
gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

Update bag.c and make again
Recompiles only bag.c

Make knows .o files come from .c files!

Makebag1a

```
bagsimple: bag.o bagsimple.o readlinep.o
    gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
$ make -f Makebag1a
cc    -c -o bag.o bag.c
cc    -c -o bagsimple.o bagsimple.c
cc    -c -o readlinep.o readlinep.c
$ gcc -o bagsimple bag.o bagsimple.o readlinep.o
$ touch bag.c
$ make -f Makebag1a
cc    -c -o bag.o bag.c
gcc -o bagsimple bag.o bagsimple.o readlinep.o
$ touch bag.h
$ make -f Makebag1a
make: 'bagsimple' is up to date
```

Problem: make does not recompile if .h files changes

We must tell make about .h files that go with .o files

Makebag1b

```
bagsimple: bag.o bagsimple.o readlinep.o
    gcc -o bagsimple bag.o bagsimple.o readlinep.o

bag.o: bag.h
bagsimple.o: bag.h readlinep.h
readlinep.o: readlinep.h
```

Tell make that .h files go with .o files

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
$ make -f Makebag1b
cc    -c -o bag.o bag.c
cc    -c -o bagsimple.o bagsimple.c
cc    -c -o readlinep.o readlinep.c
$ gcc -o bagsimple bag.o bagsimple.o readlinep.o
```

We must tell make about .h files that go with .o files

Makebag1b

```
bagsimple: bag.o bagsimple.o readlinep.o
	gcc –o bagsimple bag.o bagsimple.o readlinep.o

bag.o: bag.h
bagsimple.o: bag.h readlinep.h
readlinep.o: readlinep.h
```

Tell make that .h files go with .h files

```
$ rm *.o
rm: remove regular file 'bag.o'? y
rm: remove regular file 'bagsimple.o'? y
rm: remove regular file 'readlinep.o'? y
$ make –f Makebag1b
cc    –c –o bag.o bag.c
cc    –c –o bagsimple.o bagsimple.c
cc    –c –o readlinep.o readlinep.c
$ gcc –o bagsimple bag.o bagsimple.o readlinep.o
$ touch bag.h
$ make –f Makebag1b
cc    –c –o bag.o bag.c
cc    –c –o bagsimple.o bagsimple.c
gcc –o bagsimple bag.o bagsimple.o readlinep.o
```

Changing .h causes recompilation

We commonly add a "test" for testing and a "clean" target to remove old files

Makebag1c

```
bagsimple: bag.o bagsimple.o readlinep.o
   gcc -o bagsimple bag.o bagsimple.o readlinep.o

bag.o: bag.h
bagsimple.o: bag.h readlinep.h
readlinep.o: readlinep.h

test:
    @echo "This is a test"

clean:
    rm -f *.o
    rm -f bagsimple
```

Put testing code here to ensure same tests can be run after changes are made to program

Delete .o files and executable
-f forces delete

```
$ make -f Makebag1c test
This is a test
$ make -f Makebag1c clean
rm -f *.o
rm -f bagsimple
```

Give target name to run that target

Make provides macros that make things simpler

Makefile

```
# Makefile for the "bagsimple" program that uses the "bag" module.
#
# CS 50, Fall 2022

CC = gcc
CFLAGS = -Wall -pedantic -std=c11 -ggdb
PROG = bagsimple
OBJS = bagsimple.o bag.o readlinep.o
LIBS = -lm

.PHONY: all clean

all: bagsimple

# executable depends on object files
$(PROG): $(OBJS)
	$(CC) $(CFLAGS) $(OBJS) $(LIBS) -o $(PROG)

# object files depend on header files
bagsimple.o: bag.h readlinep.h
bag.o: bag.h
readlinep.o: readlinep.h

clean:
	rm -f $(PROG)
	rm -f *.o
```

Macro format: MACRO = value

Make knows about CC macro, will use this as the compiler for .c files

Provide our compiler flags in CFLAGS

Name of executable to produce

Dependencies of executable

Any libraries needed such as math

Tells make these targets do not produce a file (not required)

Typing make with no target runs the first one, all commonly put first so just typing "make" runs "make all"

PROG target = bagsimple
OBJS gives object file dependencies
CC tells which compiler to use
CFLAGS for compiler
LIBS gives any needed libraries to link

Same as previous

Make also has several automatic macros

Automatic macros
$@ name of the current target
$? the list of dependencies that are newer than the target
$^ the list of dependencies for this target

For example, we could rewrite our bagsimple target as

```
$(PROG): $(OBJS)
	$(CC) $(CFLAGS) $^ -o $@
```

Agenda

1. Makefiles
2. Compiling bagsimple with make
3. Activity