Recitation 4: Makefiles

- Course News
- Agenda
- Makefiles
- Quiz!

Griffin Hurt
Undergraduate Teaching Fellow
griffhurt@pitt.edu
https://griffinhurt.com

Spring 2024, Term 2244
Friday 2 PM Recitation
Feb 9th, 2024

Slides adapted from
Shinwoo Kim, Martha Dixon, and Vinicius Petrucci

Department of Computer Science
School of Computing & Information
University of Pittsburgh

Course News

Project 1 is out, due February 19th at 5:59PM

- Recommend starting early!
- Happy to help in office hours

Agenda

Makefile

- No submissions

Quiz

- Gradescope → Q1: C quiz
- It’s timed

Makefiles

Automating and Optimizing Builds

Why and goal

Multiple files can be compiled independently and then merged together in a process called **linking**.

- Generally, these two phases use different tools behind the scenes.

Project 1.

- Write a Makefile to compile multiple files.

A Brief Overview Of Makefiles

What is make?

- The make utility is a software tool for managing and maintaining computer programs consisting many component files. The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them
- Make reads its instruction from Makefile (called the descriptor file) by default.
- Makefile is a way of automating software building procedure and other complex tasks with dependencies.
- Makefile contains: dependency rules, macros and suffix(or implicit) rules.

How does it work?

- The relationships are described in a file named “Makefile” [by default]
  - You can name it differently, but it’s not current practice!
  - https://www.gnu.org/software/make/manual/make.html#Makefile-Names
- Make will look into that file, and follow the rules described

Allows us to create custom settings and compile multiple files quickly with a single command (make)

Rules

Rules specifying how to make files

- How to make a file is specified by a **recipe**
- **Target** is the file created using the recipe
- Targets have **prerequisite files**
- Prerequisites can be made by another rule

Example Rule:

```
target: prerequisites
    recipe
```

```
prog: main.o
        gcc -o prog main.o
main.o: main.c
        gcc -c main.c
```

Target main depends on `main.o` (that is created by another rule) and it's made by invoking `gcc –o prog main.o`

Because the Makefile has a rule to generate main.o…
But you still need main.c

Example: without makefile

```
int hellomake(){
        return 0;
}
```

Compile this code:

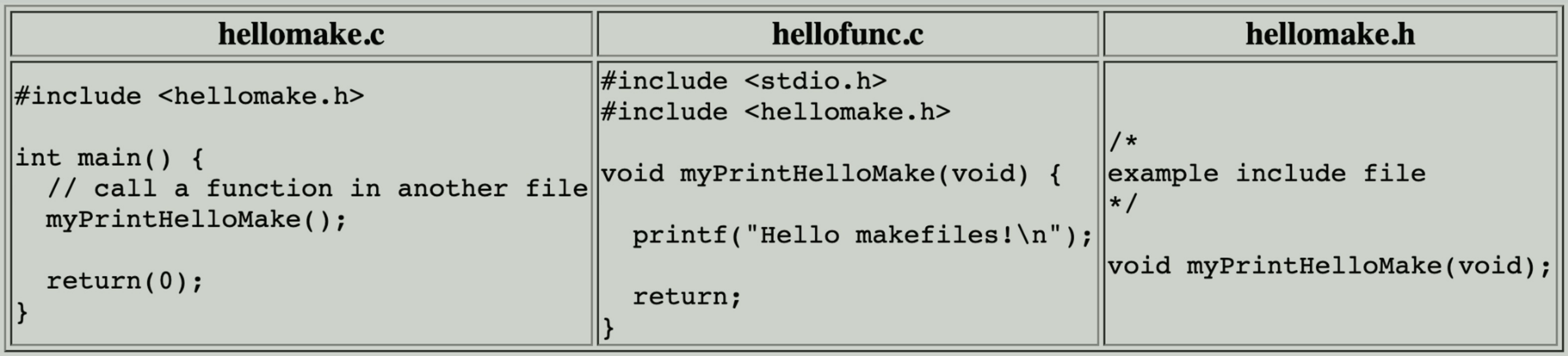
- **gcc -c hellomake.c**
- **gcc -o hellomake hellomake.o**

The "-c" argument to gcc will create a **hellomake.o object file** instead of link an entire executable.

We can now **link** the object file with the C standard library and create an executable called **hellomake** using this line.

https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Example: without makefile



hellomake.c

```
#include <hellomake.h>

int main() {
  // call a function in another file
  myPrintHelloMake();

  return(0);
}
```

hellofunc.c

```
#include <stdio.h>
#include <hellomake.h>

void myPrintHelloMake(void) {

  printf("Hello makefiles!\n");

  return;
}
```

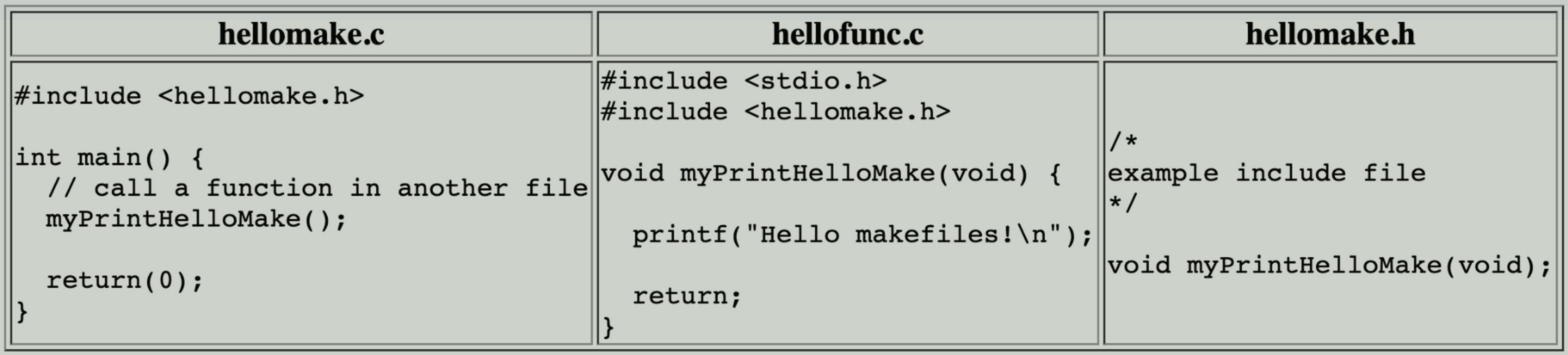
hellomake.h

```
/*
example include file
*/

void myPrintHelloMake(void);
```

```
gcc -c hellomake.c

gcc -c hellofunc.c

gcc -o hellomake hellomake.o hellofunc.o
```

https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Example: with makefile



hellomake.c

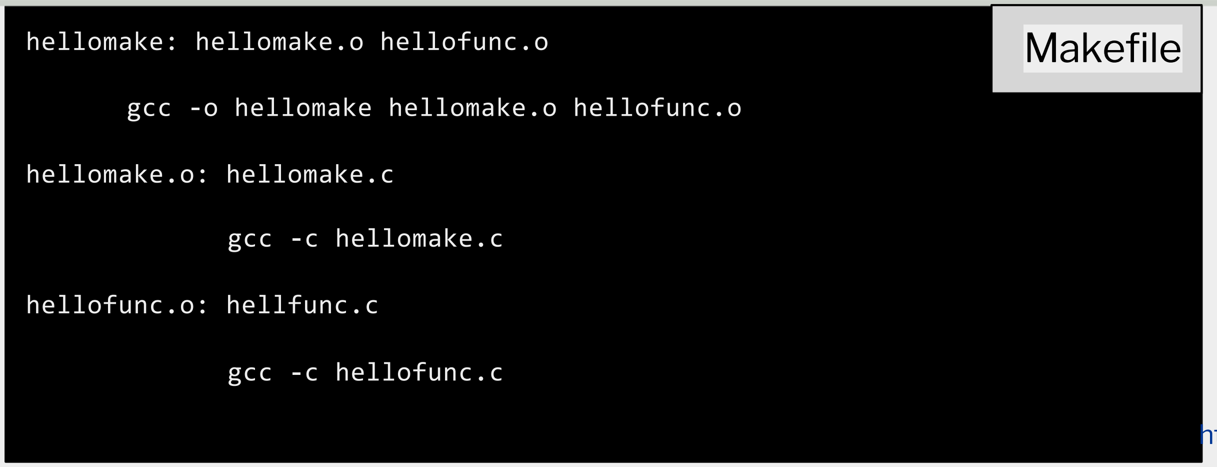
```c
#include <hellomake.h>

int main() {
  // call a function in another file
  myPrintHelloMake();

  return(0);
}
```

hellofunc.c

```c
#include <stdio.h>
#include <hellomake.h>

void myPrintHelloMake(void) {

  printf("Hello makefiles!\n");

  return;
}
```

hellomake.h

```c
/*
example include file
*/

void myPrintHelloMake(void);
```



Makefile

```make
hellomake: hellomake.o hellofunc.o

      gcc -o hellomake hellomake.o hellofunc.o

hellomake.o: hellomake.c

            gcc -c hellomake.c

hellofunc.o: hellfunc.c

            gcc -c hellofunc.c
```

Hint: Indentation are strictly tabs.

https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

More On MakeFiles

We can do a lot with MakeFiles, using custom rules and commands, variables, functions, conditional expressions, and more…

Read more about them on the course website

- https://cs0449.gitlab.io/sp2023/resources/

Official documentation

- https://www.gnu.org/software/make/manual/make.html

Quiz Time!

Password is: __________