Introduction to Computer Science and Programming in C

Session 19: November 11, 2008

Columbia University

Announcements

- Homework 3 due now
- Homework 4 is out, due last day of class: December 4 before class
- Final Exam: Tuesday, 12/16, 1:10 pm - 4:00 pm Mudd 233 (our normal room)

Review

- Pseudocode:
  - Precise like programming language
  - Understandable like English
- Headers, .h files
  - Declares global vars, functions, custom types
  - Shared between modules of large program

Today

- Modular Programming (continued)
- Makefiles

gcc -c

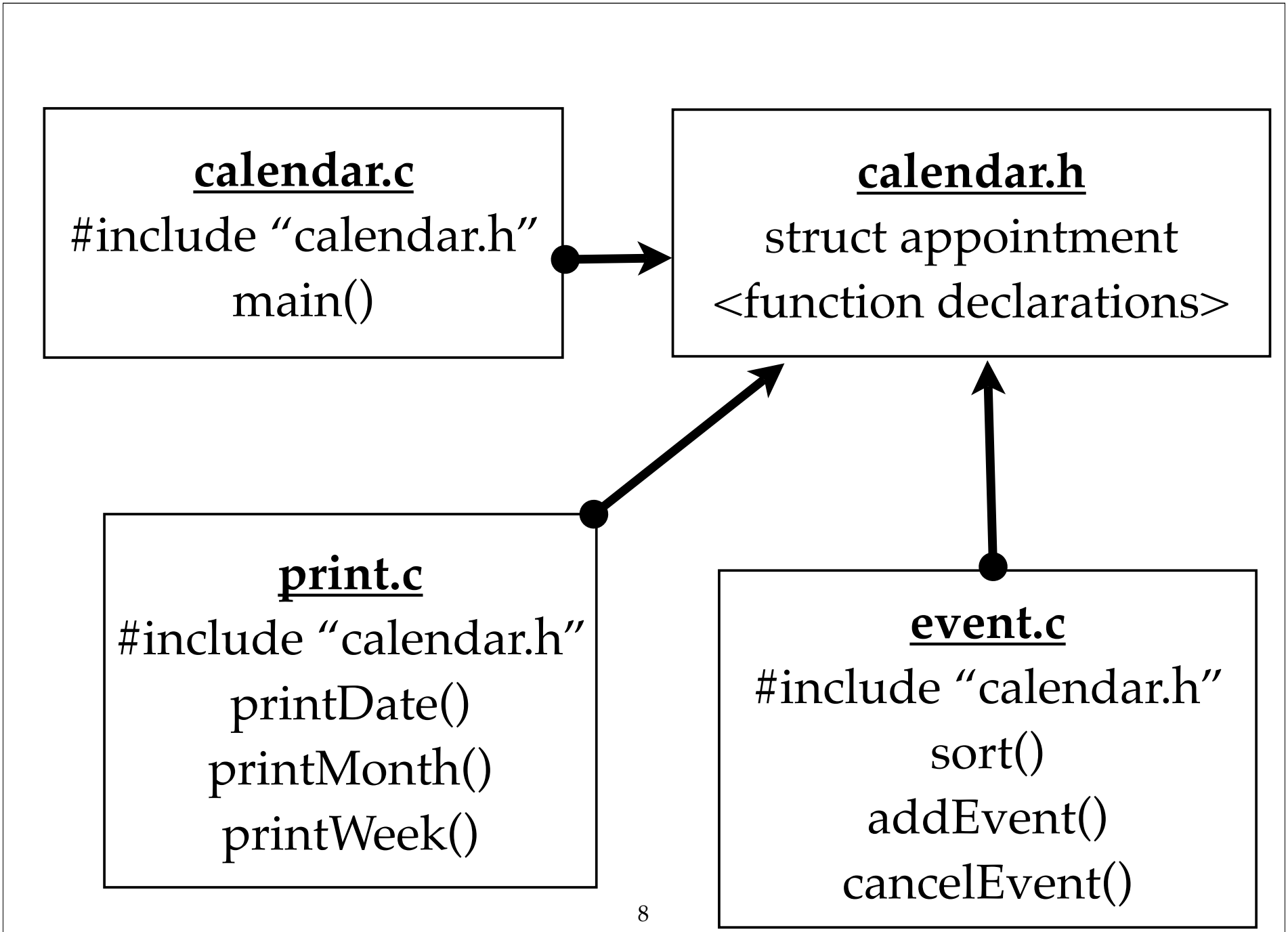
- Last time I had trouble compiling individual file without a main function
- Use “gcc -c” to compile a file as an object without a main function. Default output for file.c is file.o

Modular Programming

- **modular** - Designed with standardized units or dimensions, as for easy assembly and repair or flexible arrangement and use: *modular furniture; modular homes*.
- Organize programs into interchangeable parts
- Keep functions that deal with a certain type together, but separate them from functions that deal with other types.

calendar.c

struct appointment
sort()
addEvent()
cancelEvent()
printDate()
printMonth()
printWeek()
...
main()


calendar.c
#include “calendar.h”
main()
calendar.h
struct appointment
<function declarations>
print.c
#include “calendar.h”
printDate()
printMonth()
printWeek()
event.c
#include “calendar.h”
sort()
addEvent()
cancelEvent()

Object Oriented Programming

- Strictly modularize programs
- All variables are objects
- Computation is the interaction of objects
- All objects have “classes”

Classes

- C does not explicitly use classes, but it is useful to think in terms of classes.
- A class is a generalization of a type
- Type - what kind of information is stored
- Class - what kind of information is stored
  what we can do with this information
- A collection of variable fields and functions

C and Classes

- We can approximate OOP with C
- Put type definition (struct) and functions that work with that type in separate file
- OOP likes to set certain fields and functions public and private (whether they are visible to other objects). C can't do this.

Object Oriented Programming

- OOP is like the abolishment of **goto**
- Organizes programmers' thinking to reduce errors
- Helps programmers collaborate

Makefiles

- We use “make”, which is a compiler utility
- “make” looks for a file in your directory called “Makefile”, which contains:
  - Comments
  - Macros
  - Rules

Makefile Syntax

- Comments are indicated by a #
  # This is a comment, it won't affect make
- Macros are defined by =
  SIZE = 10
  and used with $(): **echo $(SIZE)** -> **echo 10**
- Rules, the most important part, are the compiling commands

Makefile Rules

- **target: source [source2] [source3]**
  command
- Then, typing “make target” in Unix will compile **source** using **command.**
- Make checks if **target** needs to be compiled
- If command is omitted, default command is used: $(CC) $(CFLAGS) -c **source**

Makefiles

- If target already was compiled and **source** has not changed, make will skip
- Extremely helpful when compiling multi-file code
- Macros allow programmers to customize makefiles for different systems

Example

```
#-----------------------------------------------#
#   Makefile for unix systems    #
#    using a GNU C compiler         #
#-----------------------------------------------#
CC=gcc
CFLAGS=-g -Wall -D__USE_FIXED_PROTOTYPES__ -ansi

all:   hist

hist: hist.o ia.o
    $(CC) $(CFLAGS) -o  hist hist.o ia.o

hist.o: hist.c ia.h

ia.o: ia.c ia.h

clean:
    rm -f hist hist.o ia.o
```

Reading

- Practical C Programming. Chapter 18