ECS 36A, June 4, 2024

Announcements

- Thursday June 6 is the last day of classes
- Those of you with discussion section on Friday should go to Thursday's discussion section
    - They will review for the final
- Thursday's class will begin with a review
    - So bring your questions!
- All parts of Canvas will be updated *except* the solutions to Homework 4 and Extra Credit 3
    - Those will be posted on June 12 in the morning

Announcements

- Final study guide and sample final are posted
  - Answers to the sample final are on Canvas
- Homework 4 is posted
  - Submit the program for the third one (the problem on the birthday paradox) to Canvas, *not* to Gradescope
  - Due date is June 6, last day of classes
  - *Late* due date is June 11, the day before the final exam
- Extra credit 3 is also posted
  - Submit the program to Canvas, *not* to Gradescope
  - Due date, late due date same as for homework 4

Multi-File Programs

- .c files are C source code
- .h files are header files
  - Contain macro definitions, prototypes, global variables
- Library files
  - Contain functions, variables used by many programs

C Source Files

- You've seen single file programs
- Multi-file programs

Multi-File Programs

- .c files are C source code
- .h files are header files
  - Contain macro definitions, prototypes, global variables
- Library files
  - Some linked automatically
  - Others need to be named either using –l*xxx* or lib*xxx*.a

Header Files

- Used when a program in many files needs access to data in other files
  - Example: stdio.h gives prototypes, variable declarations that are defined in the standard C library
- Do *not* define functions or variables in them!
- Example: in the header file ab.h, we have

```
int glob;
```

- but the header file is included in both a.c and b.c
- Loader gives an error (multiple definitions of glob)

`extern` Keyword

- `extern` declares a variable but does not define it
- Declaration: gives type, name of variable but does not allocate memory

```
extern int glob;
```

- Definition: like declaration, and also allocates memory

```
int glob;
```

C Source Files

- Contain C functions
- Exactly one file must have a main() routine
  - When program starts, operating system executes a small routine to set up arguments and environment variables, and then branches to main() and begins there
- Every function and variable can be defined only once, in one file
  - But it can be declared wherever it is to be used

Defining for One File Only

- Use keyword static
- Function, variable declared static can only be used in the file where it is declared
  - And after it is defined
  - Always define it when you declare it
- Useful to confine variable, function use to one file
  - Prevents accidents

Libraries

- Usually referenced as –l*xxx*
- One library automatically loaded: C library
  - Includes stdio functions, system calls, string functions, other functions
- Other common libraries:
  - –lm Math library

`make`

- Automated way to compile your program
- Very useful for compiles requiring more than 1 source file
  - If you change a source file, it automatically figures out what needs to be recompiled
- You can define variables and functions
- Can handle very large programs

Makefile (or makefile)

```
# basic one
a:   ar.o br.o
     $(CC) $(CFLAGS) –o a ar.o br.o

br.o:    abr.h

ar.o:    abr.h
```

Makefile (or makefile)

- CC, CFLAGS predefined variables
  - You can override them
- Comment lines begin with #
- Target is on left, followed by colon ":", followed by list of *dependencies*
  - **Must be separated by a tab character; spaces don't work!!!!!**
- Elements of that list may be targets themselves
  - So in the previous file, "a" is the target; any change to ar.o or br.o will cause it to be remade
  - ar.o depends on abr.h, so any change to that will cause it to be rebuilt
  - Note a .o file depends on a .c file (or some other source code filr)

Makefile (or makefile)

```
# with variables
CC = gcc
CFLAGS = -ansi –pedantic –Wall –Wextra –g

a:   ar.o br.o
     $(CC) $(CFLAGS) –o ar.o br.o

br.o:    abr.h

ar.o:    abr.h
```

Stack

- Structure in which objects are added to top, taken from top
- "push" onto stack, "pop" off stack