UNIVERSITY®

CS 240: Programming in C

Lecture 22: Large-Scale Development
Random Number Generation
Graphical Programming

Prof. Jeff Turkstra

1

Announcements

- Midterm Exam 2 tomorrow night!
- No feasting with faculty this week
- Final Exam is Thursday, 5/8 10:30am – 12:30pm
- Please be sure you're contacting the right person
  - I often don't respond to email quickly, especially if it should go to someone else

2

Why use libraries?

- The C language has no built-in functions
- You are always using a library: The C Standard Library (/usr/lib/libc.so) that contains functions like printf(), strcpy(), and similar friends
- Create your own libraries when you have a lot of object files that you need to keep organized or need to share with someone else
- Linking in a single library that contains 7,000 object files is faster than linking against 7,000 separate object files....

3

Example project

- Suppose I have a large software project that has the following data structures:
  country
  state
  county
  township
  road
- There are various interactions. E.g., a county contains a list of townships, a road may contain a list of townships that it connects, etc

4

Rule 1: Declare one data structure per file

- I might have a header file called county.h that declares a struct county:
```
struct county {
  struct township *township_array[];
  ...
};
```
- What do we do about that struct township?

5

Two ways to handle forward references...

- If a data structure is referred to only by pointer (e.g., struct township * within county), you can create a forward declaration for it:
```
struct township;
```

```
struct county {
  struct township *township_array[];
  ...
};
```
- Otherwise, you need to #include the full definition...

6

Rule #2: Use #includes in your header files...

- The other way to handle townships within a county:
  ```
  #include "township.h"

  struct county {
    struct township *township_array[];
    …
  };
  ```
- And you can guess what's in township.h

Rule #3: Use only as many #includes as you need

- Within county.h, we might #include lots of other stuff that is unnecessary:
  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <assert.h>
  #include <blahblahblah.h>

  #include "township.h"
  struct county {
    struct township *township_array[];
    …
  };
  ```
- Put these extra #includes in C files only.

Rule #4: Make sure you #include a file only once..

- What happens now if, in a C file, I say:
  ```
  #include "township.h"
  #include "county.h"
  ```
  Also #includes "township.h"
- This will create a "duplicate declaration" error
- We can use a simple and very common C pre-processor trick to avoid this

In every header file...

- township.h:
  ```
  #ifndef __township_h__
  #define __township_h__

  struct township {
    …
  };

  #endif  /* __township_h__ */
  ```
- You choose the style for the symbol that you use

Avoiding duplicate #includes

- Over in county.h:
  ```
  #ifndef __county_h__
  #define __county_h__
  ```
  If township.h was already #included, the #ifdef will make this #include benign.
  ```
  #include "township.h"

  struct county {
    struct township *township_array[];
    …
  };

  #endif  /* __county_h__
  ```

Avoiding duplicate #includes

- So, back in our .c file:

  ```
  #include "township.h"
  #include "county.h"
  ```
  #defines __township_h__

  township.h contents not re-included this time!

Random numbers

- "On two occasions I have been asked [by members of Parliament], 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question."
  -- Charles Babbage
- Computers <u>cannot</u> generate random numbers.
  - You get out only as much as you put in.
- Computers can generate pseudo-random sequences that look, to you and me, random

Basic pseudo-random numbers

- Quick example:

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

int main() {
  for (int i = 0; i < 10; i++) {
    printf("A random number: %d\n",
            random() % 100);
  }
}
```

- Generates 10 "random" numbers between 0..99

Better pseudo-randomness

- Change the seed each time it is run:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main() {
  srandom(time(0));
  for (int i = 0; i < 10; i++) {
    printf("A random number: %d\n",
            random() % 100);
  }
}
```

- Generates 10 "random numbers between 0..99

What you need to know about random number generation

- We could spend a lot of time on random number generation
- Much of it you won't understand until you take a probabilistic methods course
  - Sometimes covered in crypto courses
- To generate a number between x and y, inclusive, do this:
  number = x + (random() % (y − x + 1));
- That's it. Assume it is uniformly distributed.
- If you're really into random numbers, stop by and I'll give you more complicated stuff

Purdue Trivia

- Purdue has an extensive network of "steam tunnels"
- Connect to almost all buildings on campus
  - Power conduits
  - Fiber/networking
  - Steam
  - Chilled water
  - ...and more!
- There are motion sensors!

You are a programmer.

- 95% of the problems you might want to implement with a computer program involve very little in the way of data structures.
  - You can handle them RIGHT NOW.
- Most problems simply involve figuring out the existing API (Application Programming Interface) and writing things to use it
- Graphics programming is an example

"Simple" example using SDL

- "There's nothing remarkable about it. All one has to do is hit the right keys at the right time and the instrument plays itself."
-- J.S. Bach (when asked about playing the harpsichord)
- Examples here come from: http://libsdl.org/

SDL

- Simple DirectMedia Layer
- Games include Source Engine (Portal 2, L4D2, Counter-Strike Source, TF2, etc) and others
- Cross-platform library that provides low-level access to audio, keyboard, mouse, and graphics

Initialization

- `int SDL_Init(Uint32 flags);`
  - Call before all other SDL functions. Initializes SDL subsystems specified by flags
- `SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags);`
  - Set up a video window
  - `HWSURFACE`: Create it in video memory
  - `DOUBLEBUF`: Hardware double buffering

Input

- `int SDL_PollEvent(SDL_Event *event);`
  - Returns 1 if pending events, 0 otherwise
  - If event != NULL, populated with next event
- `SDL_Event`
  - Specifies type and information (see man page)
- `Uint8 *SDL_GetKeyState(int *numkeys);`
  - Snapshot of current keyboard state
  - Pointer to an array – indexed by SDLK_* symbols
  - 1 = key pressed, 0 = not

Graphics

- `SDL_Surface`
  - Represents areas of "graphical" memory that can be drawn to
  - See man page for fields
- `SDL_Surface *SDL_LoadBMP(const char *file);`
  - Load an image into an SDL_Surface
- `SDL_Rect`
  - Rectangular area
  - Used to define a blitting region

- `int SDL_BlitSurface(SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);`
  - "Fast blit" from source to destination
  - If src/dst NULL, entire surface is copied
- `int SDL_Flip(SDL_Surface *screen);`
  - Flip the video buffers

GTK+

- GIMP Toolkit
- Multi-platform toolkit for creating graphical user interfaces (GUIs)
- Some examples...

  https://book.huihoo.com/gtk+-gnome-application-development/cha-gtk.html

25

Points about examples

- Some variables referred to opaque data (e.g. window) whose contents were not manipulated directly but had access functions to make changes
- No new data structures here. Making large changes to the program would not involve a lot of data structure additions
- If you did want to add data structures, you know enough to do so and you could do great things...
- The data structures are not complicated
  - You can do this RIGHT NOW.

26

Love it / Hate it

- I love to write software, but I wasn't always very good at it.
- I made my own projects and learned a great deal in doing so.
- If you have project ideas, but don't know how to start them, come talk to me
- If you want to find out about crazy ideas that I don't have time to work on, come talk to me
- If you have no idea what to work on, I don't know that I'll be much help...

27

Disbelief is your greatest enemy

- The only thing standing between your ability to write useful and professional-looking software is your disbelief that you can actually do so.
- Later courses will refine your knowledge and discipline, but you may never be more practiced and capable than at this point
- If you enjoy programming, do what you can to make sure you get some exercise so that when you graduate, you have something interesting to show the world (and your potential employers)

28

Boiler Up!

29