CS 240: Programming in C

Lecture 22: Large Scale Development, Random Numbers, APIs

PURDUE UNIVERSITY

Announcements

- Homework 11 released
- Midterm 2 grades will be posted very soon!
  - Be sure to look at your score
  - Make any regrade requests promptly

Large Scale Development

- 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 it connects, etc.

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 to do about that struct township?

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…

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”
```

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”
```

- This will create a “duplicate declaration” error
- We can use a simple and very common C pre-processor trick to avoid this

Header guards

- Add these definitions to each header file:
- township.h:

```
#ifndef __township_h__
#define __township_h__

struct township {
  ...
};

#endif  /* __township_h__ */
```

Avoiding duplicate #includes

- Over in county.h:

```
#ifndef __county_h__
#define __county_h__

#include “township.h”

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

#endif  /* __county_h__ */
```

Avoiding duplicate #includes

- Over in county.h:

```
#ifndef __county_h__
#define __county_h__

#include “township.h”

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

#endif  /* __county_h__ */
```

If township.h was already #included, the #ifndef will make this #include benign

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

- Computers <u>cannot</u> generate random numbers
- Computers <u>can</u> generate pseudo-random sequences that *appear* 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);
  }
}
```

Even better pseudo-randomness

- Use /dev/urandom

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

int main() {
  FILE *rng = fopen("/dev/urandom", "rb");
  for (int i = 0; i < 10; i++) {
    unsigned int r;
    fread(&r, sizeof(int), 1, rng);
    printf("A random number: %d\n",
           r % 100);
  }
  fclose(rng); rng = NULL;
}
```

Even better pseudo-randomness

- Use /dev/urandom

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

int main() {
  FILE *rng = fopen("/dev/urandom", "rb");
  for (int i = 0; i < 10; i++) {
    unsigned int r;
    fread(&r, sizeof(int), 1, rng);
    printf("A random number: %d\n",
           r % 100);
  }
  fclose(rng); rng = NULL;
}
```

We can't set the seed

Random number generation

- /dev/urandom is “cryptographically secure”
- Both random() and /dev/urandom produce uniformly distributed random numbers
  - Every possible number has equal probability
- To get other distributions, you need to “transform” the uniform distribution
  - e.g., the normal distribution
  - We won’t cover this

Random number generation

- We could spend a lot of time on random number generation
- Much of it requires knowledge of probabilistic methods
  - Sometimes covered in crypto courses
- To generate a number between x and y inclusive, do this:

```
number = x + (random() % (y - x + 1));
```

- Assume it is uniformly distributed

APIs

- 95% of the problems you might want to implement with a computer program involve very little in the way of data structures and algorithms
  - 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

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 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

Graphics

```
int SDL_BlitSurface(SDL_Surface *src,
                    SDL_Rect *srcrect,
                    SDL_Surface *dst,
                    SDL_Rect *dstrect);
```

- “Fast blit” from source to destination
- If srcrect/dstrect is NULL, entire surface is copied

```
int SDL_Flip(SDL_Surface *screen);
```

- Flip the video buffers

SDL Example

```
#include <SDL/SDL.h>
#include <stdlib.h>

int main(int argc, char **argv) {
  /* Initialize SDL */
  if (SDL_Init(SDL_INIT_VIDEO) < 0) {
    fprintf(stderr, "Unable to init SDL: %s\n",
            SDL_GetError());
    return -1;
  }

  /* Set the surface format */
  SDL_Surface *screen = SDL_SetVideoMode(
    640, 480, 32, SDL_HWSURFACE);
  if (screen == NULL) {
    fprintf(stderr, "Unable to set video mode: %s\n",
            SDL_GetError());
    SDL_Quit();
    return -1;
  }
```

SDL Example

```
/* Fill each pixel with a color */
SDL_LockSurface(screen);
for (int x = 0; x < screen->w; x++) {
  for (int y = 0; y < screen->h; y++) {
    Uint8 *pixel = screen->pixels + y * screen->pitch + x * 4;
    pixel[0] = 0;  /* Blue */
    pixel[1] = (Uint8)(255 * (float)y / screen->h);  /* Green */
    pixel[2] = (Uint8)(255 * (float)x / screen->w); /* Red */
    pixel[3] = 255; /* Alpha */
  }
}
SDL_UnlockSurface(screen);
/* Flip the buffer! */
SDL_Flip(screen);
```

SDL Example

```
  /* Event loop -- wait for user to press Q */
  while (1) {
    SDL_Event event;
    while (SDL_PollEvent(&event)) {
      /* Close the window */
      if (event.type == SDL_QUIT) {
        SDL_Quit();
        return 0;
      }
      /* Press Q */
      else if (event.type == SDL_KEYDOWN) {
         if (event.key.keysym.sym == SDLK_q) {
          printf("Quit!\n");
          SDL_Quit();
          return 0;
         }
      }
    }
  }
}
```

GTK+

- GIMP Toolkit
- Multi-platform toolkit for creating graphical user interfaces (GUIs)
- Some examples found here:
  - https://book.huihoo.com/gtk+-gnome-application-development/cha-gtk.html

Points about examples

- Some variables referred to opaque data 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