week6

Tommy MacWilliam

tmacwilliam@cs50.net

October 17, 2011

Announcements

- great job on quiz0!
- pset4: returned

Today

- quiz0 debrief
- hexadecimal
- file I/O
- structs
- enums

quiz0

- questions?

Hexadecimal

- binary: base 2, so each digit is $\times 2^n$
  - digits are `0-1`
  - sometimes written as `0b00101`
- hexadecimal: base 16, so each digit is $\times 16^n$
  - digits are `0-9` and `A-F`
  - usually written as `0x0b39ac`

Hexadecimal

`0x42`

4	2
16^1	16^0

$$= 4 \times 16 + 2 \times 1 = 42$$

Converting to Hexadecimal

- 8 bits is a byte, 4 bits is a nibble
  - HA HA HA COMPUTER JOKES
- 1 nibble = 1 hexadecimal digit
  - because we can represent 4 bits with 16 different digits

Converting to Hexadecimal

- decimal: 40561
- binary: 1001111001110001

<table>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td colspan="4">9</td><td colspan="4">e</td><td colspan="4">7</td><td colspan="4">1</td></tr>
</table>

I/O

- short for input/output
  - aka reading and writing a file
- file represented by a file pointer, or `FILE*`

Opening Files

- `FILE* fp = fopen(‘‘/path/to/file.txt’’, ‘‘r’’);`
  - first argument: path to file
  - second argument: mode to open file
    - `r`: read
    - `w`: write (overwrite entire file)
    - `a`: append (start adding content at end of file)

Cursor

- each file has an associated cursor: where you are in the file
    - reading/writing bytes will start from the current position of the cursor
    - after reading/writing bytes, cursor will move forward

Reading

- `fgetc`: read a single character
- `fgets`: read an entire line into a string
- `fread`: read a specific number of bytes
- `fscanf`: read data in a given format
  - just like `sscanf`, except from a file this time

Writing

- `fputc`: write a single character
- `fputs`: write an entire line from a string
- `fwrite`: write a sequence of bytes
- `fprintf`: write data in a given format
  - just like `printf`, except to a file this time

File I/O

- example time!
  - `stringio.c, io.c`

Structs

- as seen in pset4
- container for variables of different types

```
struct x {
    int board[9][9];
    char label;
}
```

Typedef

- `typedef` allows you to create your own types

```
typedef int omginteger;
omginteger n = 5;
printf(‘‘%d\n’’, n);
```

Creating Structs

- without typedef

```
struct student {
    char* name;
    int score;
};
struct student tommy;
```

Creating Structs

- with typedef

```
typedef struct student {
    char* name;
    int score;
} student;
student tommy;
```

Initializing Structs

- `struct student tommy = { ‘‘Tommy’’, 100 };`
  - initialize parameters in the order they appear in the definition
- `struct student tommy = { .score = 100, .name = ‘‘Tommy’’ };`
  - initialize parameters in any order you want!

Accessing Fields

- access fields of structs with . or `->`
- `.` used for instances of structs

```
struct student tommy;
tommy.score = 100;
```

- `->` used for pointers to structs

```
struct* student tommy = malloc(
   sizeof(struct tommy));
tommy->score = 100;
```

Structs

- example time!
  - `structs.c`

Enums

```
#define MOVE_UP 0
#define MOVE_RIGHT 1
#define MOVE_DOWN 2
#define MOVE_LEFT 3
```

Enums

```
enum moves = { MOVE_UP, MOVE_RIGHT,
  MOVE_DOWN, MOVE_LEFT };
if (x == MOVE_UP)
    // wow, this makes #define look stupid
```

Practice Problems

- flip a bitmap upside-down
    - rotate a bitmap 90 degrees
- implement gedit (poorly)!