CS 222: More File I/O, Bits, Masks, Finite State Problems

Chris Kauffman

Week 6-1

Logistics

Reading

- Ch 9 (file i/o)
- Ch 11 finish (practical programming)

Exam 2 Thursday

Practice Problems posted tomorrow morning

Homework

- HW 5 due tonight
- HW 6 up by Thursday

HW 5 Due Tonight: Questions?

Problem 1: `outer_product()`

- No freeing required

Problem 3: `channel_params` allocation

- Make sure your array is sized right
- Lots of struggle with iteration; work examples, compare to your own results, look for a pattern
- No freeing necessary

Problem 4: Longest Line

- Same pattern of *find the best thing* as we have seen on several previous HWs
- Must use `fgetc()` of `fscanf(f,"%c",&in);` for this one to look at each character
- Deductions for use of `fgets()` of `fscanf(f,"%s",buf)`

Goals

- File Output
- File I/O for Structs
- Bit operations in C
- Modeling problems with finite state

Warm-up: Non-whitespace count

- Write a `main()` function
- Accepts a command line arg, named file
- Open file, process it
- Count all non-whitespace (word) characters in the file
- Use the `isspace(char c)` function:
  - True when a `c` is a space character
  - False otherwise

```
> gcc count_nonws.c
> a.out
usage: a.out filename
> a.out stuff.txt
stuff.txt has 23 non-whitespace character
> a.out other.txt
other.txt has 144 non-whitespace character
```

Writing Files

Done with `fprintf(file,format,arg1,arg2,...)`

- Works like `printf()` except first arg is a `FILE*`
- Open files for writing
- Will create files if they don't exist

```
#include <stdio.h>
int main(){
  FILE *f = fopen("myfile","w");
  fprintf(f,"Overwrite now\n");
  fclose(f);
}
```

A Note on Buffering

Operating systems try to optimize I/O operations

- Data doesn't get pushed to disk right away
- Guaranteed when `fclose` is called
- See `buffering-problems.c`
- Other ways to force writing (`fflush(file)`)
- See `buffering-flush.c`

Demonstration: `input2file.c`

- Reads characters of input
- Writes to a file
- A short way to save files

Exercise: File Copying

- Copy a file character by character to another file
- Both files named on command line
- Use the basic input loop provided in `input2file.c`

```
> gcc copy_file.c
> a.out fileA.txt copyA.txt
> cat fileA.txt
hello world!
I am a file
with stuff and everything
> cat copyA.txt
hello world!
I am a file
with stuff and everything
```

Bit Operations

Mangling bits puts hair on your chest.

Logical `&&` and `||` are AND and OR

- `int x = 12 || 10; // is 1`

Bitwise `&` and `|` are AND and OR

- `int x = 12 | 10; // is 14`

```
   1100         1101
OR 1010     AND 1010
-------         --------
   1110         1000
```

`^` is *bitwise* XOR (exclusive or)

`!` is *logical* not

`~` is *bitwise* not - flips bits

Bit Masks

- `#define` often used to establish masks: specific pattern of bits for use with computation

Bit Shifts

- $<<$ is left shift
  - $x = y << 3;$
  - Move all bits in y to the left by 4
  - Store the result in x
- $>>$ is right shift
  - $x = y >> 2;$
  - Move all bits in y to the right by 2
  - Store the result in x

```
        12345678
     y  10010011
y << 3  10011000
```

```
        12345678
     y  10010011
y >> 2  00100100
```

Demos: Show Bits and Shifting

showbits.c

- Shows the bits of integer arguments
- Demonstrates practical use of bit shifts and masks

```
> gcc showbits.c
> a.out 0 5 8 22 128 345 -7 -1
Binary                           Hex      Decimal
00000000000000000000000000000000        0 0
00000000000000000000000000000101        5 5
00000000000000000000000000001000        8 8
00000000000000000000000000010110       16 22
00000000000000000000000010000000       80 128
00000000000000000000000101011001      159 345
11111111111111111111111111111001 FFFFFFF9 -7
11111111111111111111111111111111 FFFFFFFF -1
Binary                           Hex      Decimal
```

Other Bit Examples

`bitshifts.c`

- Arguments are integer and shift
- Shows bits after a LEFT shift

`showbits_float.c`

- Trickier example involving showing the bits of a floating point number
- Must use a `union` to as bitwise ops only defined for integer types (`char, short, int, long`)

Exercise: Count Bits

- Write function that counts how bits are set in an integer

  `int count_ones(int num)`

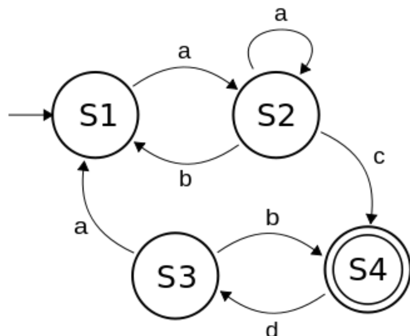
- Very helpful: loop in `showbits(int x)` function
- Provided `main()` tests

```
> gcc count_bits.c
> a.out 111
Number 111 has 6 ones
> a.out 22
Number 22 has 3 ones
> a.out -1095
Number -1095 has 28 ones
```

```
int count_ones(int num){
  // Your code here
}
int main(int argc, char **argv){
  if(argc < 2){
    printf("usage: %s integer\n",
           argv[0]);
    return -1;
  }
  int number = atoi(argv[1]);
  int ones = count_ones(number);
  printf("Number %d has %d ones\n",
         number,ones);
  return 0;
}
```

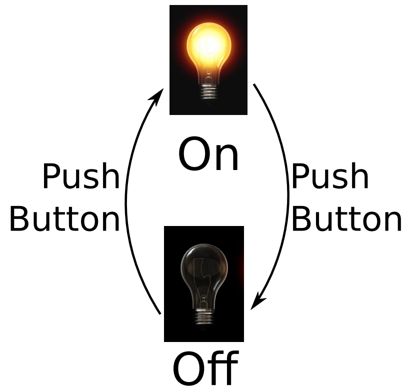
Finite State Problems

- Class of problems
- Limited (finite) number of states in which a system can be in
- Transitions from one state to another are well-defined
- Often occur with devices, small electronics, games
- Usually draw states in a map-like fashion



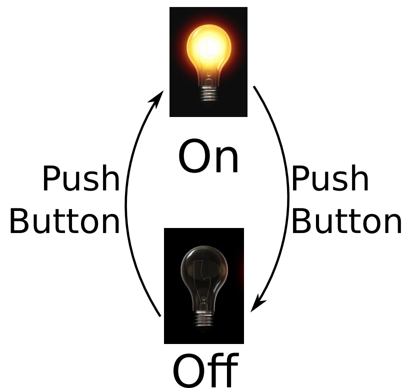
Example: The Light Switch

- Single button, push toggles light on/off
- Button: physical device that can be pushed
- Light: can be set to ON or OFF



Light Switch Code: `light_switch_easy.c`

- On starting, microcontroller sets variables corresponding to hardware
- Also runs an `init()` function which allows programmer to set their own variables
- Microcontroller runs an `update()` function every so often
- Code checks special global variables to detect button pushes
- Code sets special global variables to change lights on/off



Variant: Access Hardware State via Bit Operations

- Sometimes single bits are used to indicate hardware state
- Masks become useful for detecting and setting hardware features

Example

`PORT` global variable controls light and indicates button pushes

- Bit 0 can be written or read; turns light on and off
- Bit 1 can only be read, indicates a button was pushed

Hardcore

- `light_switch_hardcore.c`
- Uses hex values for masks

Readable

- `light_switch_readable.c`
- Uses `#define` to establish masks

Discussion: On and Off Switches

- Two buttons: On/Off
- When in ON state, light on, pushing On Button does nothing
- When in OFF state, light off, pushing Off Button does nothing
- Transition between states

Modify the code for `light_switch_readable.c` to accomodate changed model