pset3: Fifteen

Tommy MacWilliam

tmacwilliam@cs50.net

September 25, 2011

Today's Music

- Ke$ha
  - Dancing with Tears in my Eyes
  - We R Who we R
  - Kiss N Tell
  - Blow

Today

- `generate`
- Makefiles
- `find`
- `fifteen`
  - `init()`
  - `draw()`
  - `move()`
  - `won()`

TODO

1. comment `generate.c`!

Generate

- `./generate n [s]`
  - `n`: number of random numbers to generate
  - `s`: seed value

rand

- `generate` uses a pseudo-random number generator (`rand()`)
  - generates a random sequence of numbers given a seed value
- same seed? same sequence of numbers
  - helpful for debugging!

Piping

- `program > file`
  - send the output of `program` to a file called `file`
- `program < file`
  - send the contents of `file` to the input of `program`
- `program1 | program2`
  - send the output of `program1` to the input of `program2`

Piping Examples

- `./generate 1024 > numbers.txt`
  - write the output of `generate` to a file called `numbers.txt`
- `./find 13 < numbers.txt`
  - use the contents of the file `numbers.txt` as input to `find`
- `./generate 1024 | ./find 13`
  - send the output of `generate` to the input of `find`

Makefile

- specify what happens when you `make something`
- `make` will look for a file named `Makefile` in the current directory

```
find: find.c helpers.c helpers.h
    gcc -ggdb -std=c99 ... -o find find.c
```

Makefile


```
name_of_target: files we need to use
    command_to_run
```

- not just for compiling code!

```
clean:
    rm -f *.o a.out core find generate
```

Makefile

- Makefiles require **tabs**, not spaces
  - gedit in the appliance will default to spaces!
- we've provided you Makefiles, so no need to edit

TODO

1. implement sort
2. implement search

find.c

- prompts the user for numbers to fill the haystack
  - Ctrl-d tells `find` to stop asking
- then, searches haystack for the given needle
  - calls `sort` and `search`, defined in `helpers.c`

helpers.c

- `sort`: sorts the `values[]` array
  - `n` is the size of `values`
- `search`: returns `true` if `value` is found in `haystack`, else `false`
  - `n` is the size of the `haystack` array

sort

- sort `values[]` destructively
  - when `sort` returns, the array passed as an argument will be changed
  - possible because arrays are passed by reference
    - more about pass by reference this week!
- do NOT return an array (since type is `void`)

Bubble Sort

- iterate over list, swapping elements in the wrong order
  - elements “bubble” to their correct position with each iteration
- once no elements have been swapped, list must be sorted!

Bubble Sort


```
while elements have been swapped
  swapped = false
  for i = 0 to n - 2
    if array[i] > array[i + 1]
      swap array[i] and array[i + 1]
      swapped = true
```

Bubble Sort

5 0 1 6 4

Bubble Sort

0 5 1 6 4

Bubble Sort

0 1 5 6 4

Bubble Sort

0 1 5 6 4

Bubble Sort

0 1 5 4 6

Bubble Sort

0 1 5 4 6

Bubble Sort

0 1 5 4 6

Bubble Sort

0 1 4 5 6

Bubble Sort

0 1 4 5 6

Selection Sort

- build sorted list one element at a time
- start at beginning of list, find smallest element
- swap smallest element and first element
- move to second element, find smallest element, swap with second
  - no longer need to look at first element, since we know it's sorted!
- continue for every element

Selection Sort


```
for i = 0 to n - 1
  min = i
  for j = i + 1 to n
    if array[j] < array[min]
      min = j
  if array[min] != array[i]
    swap array[min] and array[i]
```

Selection Sort

5 0 1 6 4

Selection Sort

0 5 1 6 4

Selection Sort

0 1 5 6 4

Selection Sort

0 1 4 6 5

Selection Sort

0 1 4 5 6

TODO

1. ~~implement sort~~
2. implement search

search

- currently implemented as a linear search
  - does not require array to be sorted, which is why `find` works fine
  - $O(n)$, slow!
- need to implement as binary search
  - $O(\log n)$, fast!

Binary Search


```
while length of list > 0
  look at middle of list
  if number found, return true
  else if number is too high, only consider
  left half of list
  else if number is too low, only consider
  right half of list
return false
```

Binary Search

50 61 121 124 143 161 164 171 175 182

Binary Search

164 171 175 182

Binary Search

161 164

Binary Search

- can be done iteratively or recursively
  - iterative: keep moving left and right bounds
  - recursive: keep calling `search`, but with different parameters each time
    - more about recursion this week!
- in both cases, need to determine middle element and which half to cut off

TODO

1. ~~implement sort~~
2. ~~implement search~~

TODO

1. `init()`
2. `draw()`
3. `move()`
4. `won()`

Where are we?

- `main()` written for you in `fifteen.c`
  - accepts/parses command-line argument
  - creates board
  - checks if game is won and exits accordingly
  - gets input, calls move tile function

init

- `int board[DIM_MAX][DIM_MAX];`
  - 2D array representing board state
- size of board given by `d`
  - board array potentially larger than actual board (stupid C can't resize arrays)

init

- `board` needs to contain starting state of board
  - `board[x][y]` could contain element at `(x, y)`
  - `board[x][y]` could contain element at row `x` and column `y`
- board starts off in descending order
  - if number of tiles is odd, swap 2 and 1

init

- `board` also must contain the blank tile
- however, `board` must contain only `int`s
  - choose some `int` value that will never appear on the board
  - `#define`!

TODO

1. ~~`init()`~~
2. `draw()`
3. `move()`
4. `won()`

draw

- need to output the current state of the board
- remember, `board[i][j]` gives the value of the tile
  - what `i` and `j` mean is up to you!
  - make sure to print tiles in the right order!

draw

- only `printf("\n")` at the end of a row
- `printf` spaces between columns
- `printf("%2d", 5);` will print blank spaces before number if number is fewer than 2 digits

TODO

1. ~~`init()`~~
2. ~~`draw()`~~
3. `move()`
4. `won()`

move

- moving a tile is as simple as changing the `board` array
- however, not all moves are legal!
  - blank tile must be next to tile to move

move

- `move` accepts the number of the tile to move, not its position
  - need to find tile's position by searching
- also need to determine where blank tile is
  - do we need to search for it on every move, or can we just remember where it is?
- if positions are adjacent, then values in `board` can be swapped

TODO

1. ~~`init()`~~
2. ~~`draw()`~~
3. ~~`move()`~~
4. `won()`

won

- `won()` checks if the game has been won, and returns a boolean
- game is won when tiles are in increasing order
  - first tile is a 1, second tile is a 2, etc.

won

- need to iterate over `board` array and check each value
  - make sure to look at every value in a row before moving on to next row
  - make sure to look at rows in order
- if any value is incorrect, then game cannot be won

TODO

1. ~~`init()`~~
2. ~~`draw()`~~
3. ~~`move()`~~
4. ~~`won()`~~