Introduction to Computer Science and Programming in C

Session 15: October 23, 2008

Columbia University

Announcements

- Homework 3 is out. Due November 6th before class
- Everybody check your homework 2 submission files. If something is wrong with your tar file, go to office hours and get help submitting.

Today

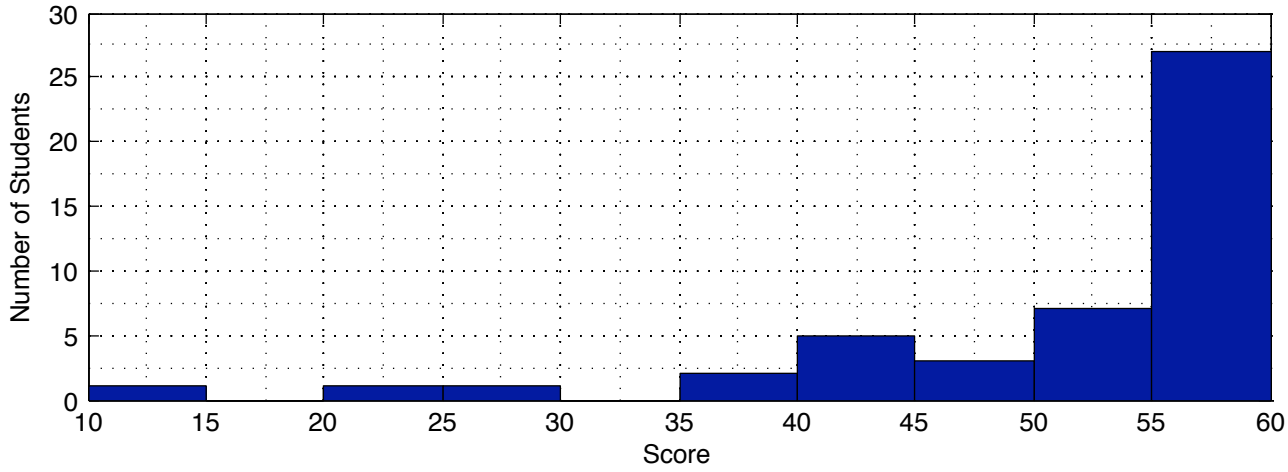
- Midterm solutions
- Pointers
- Return midterm at end of class

About Grading

- If you're struggling:
  - I will reweight your grades if you perform better at the end of the semester
  - I'll try to make your final grade reflect how well you know the material at the end of the course

Midterm Statistics

- Mean = 51.6, Median = 56, Std = 10.9 (out of 60)
- Mean = 86%, Median = 93%, Std = 18%



Variables Revisited

- What actually happens when we declare variables?

```
char a;
```

- C reserves a byte in memory to store **a**.
- Where is that memory? At an **address**.
- Under the hood, C has been keeping track of variables and their addresses.

Pointers

- We can work with memory addresses too. We can use variables called **pointers**.
- **pointer:** an address variable
- All pointers are the same size, regardless of what they point to

Pointer Operators

- Declaring a pointer variable:
```
int * x_ptr; /* declares a pointer to an int */
```
- The & operator means “the address of this thing”
- The * operator means “the thing this points to”

& and *

- ```
  int * x_ptr; /* declares a pointer to an int */
  int x, y;
  ```
- ```
  x_ptr = &x; /* set x_ptr to the address of x */
  ```
- ```
  y = *x_ptr;  /* set y to whatever x_ptr points to */
  ```
- ```
  /* is equivalent to */
  y = x;
  ```

sscanf()

- ```
  fgets(input, sizeof(input), stdin);
  sscanf(input, "%d", &index);
  ```

- This tells sscanf() the address of the variable **index** so it knows where to stick the result.
  ```

*argv[]

- `int main(int argc, char *argv[])`
- *argv[] is a bunch of strings, but argv[] (without asterisk) is an array of pointers.

FILE pointers

- When we use the FILE variables from stdio.h, we always use pointers
- `FILE *inFile = fopen(“data.txt”, “rw”);`
- fopen() returns a pointer to a file
- Then when we want to write to that file,

  `fprintf(inFile, “Hello World”);`

  We give fprintf() an address

Example

- We've talked about using arrays of structures: struct business fortune[500];
- What if we need to reorder these businesses?
  - Move large structs around
- Instead, store an array of pointers
  - Reorder the pointers, which are each relatively small.

Memory Management

- Using pointers, we are able to work with variables without explicitly naming each piece of memory
- But then we have to carefully manage what memory we have allocated and where it is stored
- This allows us to start using variable-size arrays

Reading

- Practical C Programming, Chapter 13