Slide 1

Administrivia

- Homework 5 on Web; due next week.

Slide 2

Dynamic Memory and C

- With the C89 standard, you had to decide when you compiled the program how big to make things, particularly arrays — a significant limitation.
- Variable-length arrays in C99 standard help with that, but don't solve all related problems:

  In most implementations, space is obtained for them on "the stack", an area of memory that's limited in size.

  You can return a pointer from a function, *but* not to one of the function's local variables (because these local variables cease to exist when you return from the function).

Slide 3

Dynamic Memory and C, Continued

- "Dynamic allocation" of memory gets around these limitations — allows us to request memory of whatever size we want (well, up to limitations on total memory the program can use) and have it stick around until we give it back to the system.

  (The trick here is that most implementations differentiate between two areas of memory, a "stack" used for local variables, and a "heap" used for dynamic memory allocation. Usually the former is more limited in size.)
- To request memory, use `malloc`. To return it to the system, use `free`. (For short simple programs you can not bother with `free`, but for longer and more complicated programs, you should clean up when you can, or eventually you may run out of memory.)
- Compare/contrast with Java — allocate space for objects with `new`, no explicit deallocation, garbage collection.

Slide 4

Dynamic Memory and C, Continued

- Examples:

```
int * nums = malloc(sizeof(int) * 100);
char * some_text = malloc(sizeof(char) *
20);
free(nums);
```

- Book recommends "casting" value returned by `malloc`. Other references recommend the opposite! But you should check the value — if `NULL`, system was not able to get that much memory.

Slide 5

Function Pointers

- You know from Java that there are situations in which it's useful to have method parameters that are essentially code (e.g., GUI listener methods, `compareTo` method for sorting, `run` method for threads).
- In Java, you often do this by way of a class whose main or only purpose is to hold the needed code.
- In C, however, you can explicitly pass a pointer to the function.

Slide 6

Function Pointers in C

- The type of a function pointer includes information about the number and types of parameters, plus the return type.
- Example — last parameter to library function `qsort` (in its `man` page). Call this by providing, in your code, a function with declaration

  ```
  int my_compare(const char *, const char *);
  ```

  and using `my_compare` as the last parameter to `qsort`.
  (See `sort-improved.c` on sample programs page.)
- (Other example use in *C In A Nutshell*.)

Example — Revised Sort Program

- Change the program to allow specifying at runtime that N inputs are to be generated.
- Notice also use of library function `qsort`.

Minute Essay

- None — sign in.