

Slide 1

### Administrivia

- Reminder: Homework 5 due today.
- Homework 6 on the Web. Due two weeks from today. This one is not trivial but I think gives you useful practice working with pointers.

Slide 2

### Minute Essay From Last Lecture

- More people were using the online tutorial (8) than the textbook (4), and a few used both (2).
- Most people — but not all! — said whichever one(s) they were using was helpful. Good to hear.
- A couple of people mentioned other online sources. I'll say again that while some are quite good, for C my experience is that some are decidedly not. Be cautious?
- One person who used both mentioned that with the online tutorial you can search. In days past people would cite this as an advantage by saying "you can't `grep` a dead tree". Geek humor??

## 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, added in C99 standard, help with that, but don't solve all related problems:

### Slide 3

In many 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).

## 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.

### Slide 4

(Most implementations allocate this memory “on the heap”, which is (usually?) limited only by how much total memory the program is allowed to use.)

- To request memory, use `malloc`. To return it to the system, use `free`.  
(For short simple programs you can skip this, but not good practice, since in “real” programs you may eventually run out of memory.)
- Python and Scala hide most of this from you — allocating space for objects is automatic/hidden, and space is reclaimed by automatic garbage collection.  
Makes for easier programming but possibly-unpredictable performance.

## Dynamic Memory and C, Continued

- Simple examples:

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

Slide 5

though it's better style/practice to write

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

- Some books/resources recommend "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.

- (Example — "improved" sort program.)

Slide 6

## Function Pointers

Slide 7

- You know from more-abstract languages that there are situations in which it's useful to have method parameters that are essentially code. Some languages make that easy (functions are "first-class objects") and others don't, but almost all of them provide some way to do it, since it's so useful — e.g., providing a "less-than" function for a generic sort.
- In C, you do this by explicitly passing a pointer to the function.

## Function Pointers in C

Slide 8

- 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 void *, const void *);
```

and using `my_compare` as the last parameter to `qsort`.
- (Example — "improved" sort program.)

### More vim Tips

Slide 9

- To edit multiple files at once, `vim` followed by their names. `:next` takes you to the next file, `:rew` back to the first one. `:q` exits only from the current file; `:qall` to exit from all.  
Or use “split the screen” (`:split`) to show two files (or two parts of the same file) at once; control-W twice switches between them. `:split` followed by filename splits the screen and puts the other file in the new “window”.
- You (probably? maybe?) know about `diff` to compare contents of two files. What you might not know about is `vimdiff`, which shows files side by side (or one above the other with `-o`) using colors to highlight differences.
- If you don't like the colors, there are options: Type `:colorscheme` and a space and press “tab” repeatedly to cycle through choices, enter to try one. If you find one you like, put command in `.vimrc` file.

### Minute Essay

Slide 10

- Anything noteworthy to report about Homework 5?