Dynamic Memory and Structures

11-11-2002

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment? Remember that the way it is being done now it counts for twice as much as other assignments so you want it to be done perfectly.

Problems with Memory Management

- Dynamic memory is great in many ways. It gives us a lot of flexibility that we don't have without it. However, it has some serious hazards that come along with its use.
  - Memory leaks - If you don't free memory that you allocate you eventually run out.
  - Dangling pointers - If you try to use memory that has been freed you can cause big problems.

Dangling Pointers

- A dangling pointer is a pointer to memory that you should no longer be using. You get them when you free a block of memory, but you don't destroy the pointer to it and later you try to use that memory. This can cause bugs that are very difficult to track down.
- To prevent this, you should always store NULL in a pointer after freeing it unless it is going to pass out of scope immediately.

Structures in Memory

- Last time we talked about structures, but we didn't really discuss what they look like in memory.
- As you might expect, a struct simply gets a chunk of memory big enough to hold all its members.
- What the dot notation does is to add the proper offset to the address of the structure to get the member with that name.

Pointers to Structures

- When we have a pointer to a struct we have to dereference it to get to the members in that struct.
- There is a shorthand notation for this that uses an arrow, ->. This is just a shorthand for a dereference and a dot.

```
var->member ≡ (*var).member
```

- This is helpful in many situations. Since we often pass pointers to structs you might see this more than a plain dot.

Arrays of Structures and Arrays in Structures

- We used this in code last time, but we should be explicit with how we use these.
- For an array of structures the brackets need to go by the variable name, before the dot/arrow (if there is any). Remember the offset is the size of the type.
- When the member is an array the brackets go after the member name.

2-D Dynamic Arrays

- We can make multidimensional arrays in C by simply having pointers to pointers [to pointers [to pointers …]].
- For 2-D, in memory we have a pointer to a dynamic array of pointers and each of those points to an address with an array of the proper type.
- Note that this can create non-rectangular arrays. Managing lengths can be a problem.

Minute Essay

- What questions do you have about today's material? What is unclear to you about the use of structures or dynamic memory?
- After today we switch gears for a bit and talk about the string libraries then sorting and searching.