CS2261 Media Device Architecture

* Started class with Quiz
* OO will be focusing more on teaching us C programming concepts applicable in real world situations while recitation will focus on our final project
* TA Applications are open

Notes:

Write a function that returns the length of an array:

* Recall the sizeOf function. Can be used to write a single line function

Write a function that returns the duplicate of an array (returns an int\*) and takes in an int \*

* The issue faced here is that any variable created within the function will end disappear as soon as the function ends.
* Can’t make the duplicate array variable static, you’ll get an error because size requirements are unknown
* Solution? **Dynamic Memory Allocation**

Dynamic Memory Allocation

* Used when space requirements are unknown at compile time
* Create data, within a function, that should live beyond calls of that function
* EXRAM is where the dynamic memory functions allocate to.
  + Heap lives here
* Function: malloc(size\_t n)
  + Ask for space during runtime, get a pointer to space of heap (if request denied, then you’ll get a NULL), put what you need in the space and return the space

Malloc

* Prototype: void \* malloc(size\_t n);
  + Return type is void\* to because void\* is the generic type, allows the space you asked for to hold any type of data
* size\_t
  + Built-in typedef/alias for some integer type
  + Could be int, long, or something else depending on system
  + Use SIZEOF if necessary for this argument
    - Example: need space for 25 integers? Get the size of 25 ints by putting in 25\*sizeOf(int)
    - Or, even better, use pointers:
      * double \* ptr;
      * ptr = malloc (n\*sizeOf(\*ptr));
* If there’s memory, you get a pointer to location
  + If not (e.g. user data is too big), null is returned
  + Always check for null and act accordingly
    - Abort
    - Ask again
    - Save user data
    - Ask for less
    - Free up something
* Immediately after requesting space, check for a NULL error
  + if (NULL == ptr) { /\*ptr is set to NULL \*/
    - /\* HANDLE ERROR \*?
  + }

Free

* Return the space for future malloc calls
* Prototype: void free(void \*)
* Call: free(ptr)
  + Freed memory is not cleared
  + Just notifies C that it can reuse the space in future malloc calls
  + no erasing or nulling of pointers and space is done
  + free() does not actually change ptr
* free needs the exact original address to work properly

Changing space requirements (realloc)

* say you need more or less memory than you originally asked for.
* Need to free up or get more memory while retaining whatever you already stored
  + Can copy memory over to another space requested by malloc
  + Or use realloc
* Realloc
  + Allows us to shift our allocation decision
  + Mallocs space for the new block if that fails it returns NULL otherwise...
    - Copy all the data in the old block into the new block
    - Free up the old block (note the data would still be there)
    - Return the pointer to the new block
  + Prototype: void \* realloc (void \* ptr, size\_t n);
  + Arguments: pointer to the old position and the new size we want
  + Returns the pointer to the new space
  + 2 special cases:
    - if you pass in null, it acts just like malloc
    - if you pass 0 bytes for n, it acts just like free
  + There’s always the possibility of getting null returned if realloc can’t find more space for you
    - This can cause the issue of losing your original pointer if you use realloc without a backup variable to keep track of your original pointer
      * Memory Leaks

Memory Leaks

* Memory leaks occur when the programmer loses track of memory allocated by malloc or other functions that call malloc
* Obviously a program that runs for a fraction of a second and then terminates will not cause a problem if it leaks a few bytes of memory.
* However, for real world programs memory leaks are not acceptable.

Dynamic Allocation: What can go wrong?

* Allocate a block of memory and use the contents without initialization
* Free a block but continue to use the contents
* Call realloc to expand a block of memory and then once moved continue to use the old address
* Allocate a block and lose it by losing the value of the pointer
* Read or write beyond the boundaries of the block
* FAIL TO NOTICE ERROR CONDITIONS

Manage your own memory