! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

note @566 stop following 109 views

Actions

# MALLOCLAB: Tips & stuff I've learned about

Hello all! I hope the final assignment is going well. Sorry I didn't get this out sooner. I don't have much for this assignment because it's mostly about keeping everything straight and paying close attention to details.

It's been great working with you all.

**Intro**

What's cool about this lab is that it highlights the difference between C/C++ and most other programming languages: you have wide control over memory management. The function `malloc` is the C version of C++'s `new`.

You've used this in cachelab. The following two lines are equivalent for declaring an integer array.

```
int* array = new int[size];
int* array2 = malloc(size * sizeof(int));
```

**Tips**

- Pay close attention to when you have variables that indicate the size of the payload vs. the size of the allocated block, including the header and footer (8 additional bytes). You will almost always be working with the size of the allocated block, except for dealing with the user's input, where the user will specify the size of the payload. In the previous example, the size of the payload is `size`, while the size of the allocated block is `size + 8`.
- The same goes for pointers to the payload vs. pointers to the allocated block. The user works with pointers to the payload, and you will work with pointers to the allocated block. If you have a pointer to the payload (given by `free` most likely), you can subtract 4 bytes to get the pointer to the allocated block, and then remove it from the free list if needed.
- The first way to implement `realloc` for most people is to reduce fragmentation. This is done by performing a coalesce within `realloc`, based on the size of the current block, and whether the previous and/or next blocks are free. If you're moving a block to the start of the previous free block, or moving it to a completely different place in memory altogether, you have to take care for the address you're returning: it's going to be different than the one you received. Make sure to remove and add blocks to the free list after you perform your reallocation. Play around with which of the cases you check first.

**GDB**

Use `gdb ./bin/...` to run either a test or a specific trace. Make sure you navigate to the executable. Then you can set breakpoints either in your code or in one of the test files, using `break mm.c:150` for example.

Make sure the size of you allocated blocks are what you expect them to be. You can call functions inside gdb, so for example you can keep printing `print mm_block_next(mm_block_next(...(mm_block_next(addr))))` as long as you've taken note of the first block you've allocated / start of heap (printed from gdb or saved it in a global variable for example) . You can then use `print mm_block_size(...)` to make sure your block sizes are matching expected sizes.

You can also walk through your free list using `mm_list_next(...mm_list_next(mm_list_headp))` , printing sizes if you need as well.

*Thank you to the staff and students for being awesome. Comments and/or critiques are encouraged, and I'd love to see you in my OHs!*
*Written by Anthony Winney.*

malloclab

Edit good note | 1 Updated 5 months ago by Anthony Winney

**followup discussions** *for lingering questions and comments*

Start a new followup discussion

Compose a new followup discussion