Software Systems

Day 13 - malloc

Dynamic Memory Management

- In the reading, you learned a bit about dynamic memory management.
- In some programs, you don't know in advance how much memory you will need - for these situations, you can use the heap.
- Functions like malloc and free let you manage memory on the heap.

Dynamic Memory Management

- malloc returns a value of type void* (a generic pointer).
- This represents a pointer to some unknown type of value, so you have to cast it to something first:
```
char* string = (char*)malloc(42);
```
- malloc takes number of bytes, so you typically use sizeof to allocate space for something like an array:
```
int* array = (int*)malloc(42*sizeof(int));
```

Dynamic Memory Management

- Exercise:
  - Pull from upstream if you haven't already.
  - In the class-sessions folder for today, there is a folder called shapes.
  - Take a quick look at the structs in src/rectangle.h.
  - In src/main.c there, implement the main function to allocate an array of 5 points using malloc.
  - Print out the pointer (%p) you get for that array.
  - Then print each of those points. What do you get?

Dynamic Memory Management

- Dynamic memory does not automatically go out of scope like stack memory does.
- If you don't clean up allocated memory, you can cause a memory leak.
- At the end of the program, all heap memory is freed anyway - but memory leaks can be disastrous for long-running programs (games, servers, etc).
- Use free to release the memory held by a pointer.

Dynamic Memory Management

- Exercise:
  - Install valgrind on your machine: sudo apt install valgrind (or similar).
  - Go into build/src in the shapes folder, and run
    valgrind --leak-check=full ./main
    What do you get?
  - Add a call to free at the end of your main function.
  - Then run valgrind again. What do you get?

Dynamic Memory Management

- There are a two other functions that are helpful to know about:
  - calloc allocates and zeros out a chunk of memory.
  - realloc will resize your memory, potentially allocating new memory.
- If you realloc, **you cannot use the old pointer**.

Dynamic Memory Management

- The malloc API can be rather unforgiving, so here's a few things to check over if you run into memory issues:
  - Are you using memory that hasn't been allocated yet?
  - Are you using memory that has already been freed?
  - Have you freed memory that hasn't been allocated yet?
  - Are you freeing the memory more than once?
  - Are you reallocating memory before it's been allocated or after it's been freed?
- It can be surprisingly hard to keep these rules in large software, particularly with make/free functions.

malloc

- Memory leaks are particularly nasty because you won't notice them for a while.
- And even if you do, the program doesn't always crash - it might just get really slow because it keeps swapping new memory into RAM.
- Some tips:
  - **Every `malloc` needs a corresponding `free`**.
  - If you have a `make_X` function, have a `free_X` function.
  - Document your memory interface so you know what memory the library handles and what memory you need to handle.
  - **Every piece of memory needs one owner** to handle `malloc` and `free`.

Dynamic Memory Management

- Exercise:
  - In shapes/src, the rectangle.c function contains incomplete implementations of the functions in rectangle.h. Fill these in.
  - Rewrite main to use these functions.
  - Check that there are no leaks with Valgrind.

Open Project Time

- This is the home stretch for projects.
- Use this time to work together or get help from the course staff.