C Memory Model

Boxes and Arrows

C inherits its memory model from the operating system. In virtually-addressed operating systems like UNIX and Windows, the view is of a *single contiguous space* of memory, addressed at byte offsets, typically from address `0x0` to a large address that the kernel decides is "big enough." You can think of this space as like "one gigantic array." Unlike most other programming languages, a C program can see all the data in its address space: not just the call stack and heap, but constants, static data, program text, and other data structures that C itself uses for bookkeeping.

C lets you inspect *and modify* nearly all of the data you can address. Although different platforms impose various additional restrictions, in general, if you can observe it, C lets you mangle it. As a result, C programmers need to be exceedingly careful about *where* and *for how long* they store their data.

On most modern platforms, code and static data are stored beginning from the lowest addressable memory. The heap begins after code and static data and "grows upward." The call stack begins at the highest addressable memory and "grows downward." In between the heap and stack is an empty region into which either data structure can grow.

In this class, we use a simple model to help understand how C's memory features work. Once you have internalized these rules, you will find reading and writing C programs to be much easier, and you'll be a pro at spotting memory bugs. This model is *visual*. As you simulate a program, *draw as instructed* on a piece of paper.

The Rules

1. Initialize diagram with empty stack and heap.
2. When a function is called, put a box on the stack, and label it with the function's name.
3. Put global variables outside the box.
4. Put local (automatic) variables inside the box, including function parameters.
5. Manage allocated variables on the heap.
   (a) `malloc` adds objects.
   (b) `free` removes objects.
6. As the function runs, update values.
7. Returning from a function pops the stack frame and, if the function returns a value, assigns it to the storage awaiting the return value.