Intro to the Memory Model

Memory Awareness

We will discuss a "memory model" to help you visualize the memory associated with a running C program. This is helpful because:

- C is not memory safe
  - It allows the programmer to directly access (read/write) virtual memory via the address operator & and pointers
    - Simple, common mistakes are often ignored by the compiler but wreak havoc (often silently) during run time
- C programs must do their own memory management
  - allocation: memory is requested with malloc(). It's up to the programmer to organize it. (vs. Java's new operator, objects)
  - deallocation: must explicity free() unneeded memory that was allocated (vs. Java garbage collection)

A Memory Model

- Here is one abstract view of the memory model of a C process
- Actual models differ depending on compiler, processor, etc.. For example the stack may either grow up or down. (Here, the stack grows up and the heap grows down.)
- DATA: stores compiled code, global variables, and "static variables" (discussed later)
- HEAP: stores "dynamically allocated" memory (discussed later)
- STACK: holds "stack frames" or "activation records" for each function call. Stores function parameters, local variables, program counter, return value (if any).

The Stack grows, shrinks

```
int a(int);
int b(int);

int num = 500;

int main() {
  int num = 10;
  double d = .5;
  num = a(5);
}

int a(int num){
  int y = b(num);
  num *= 2;
  return y;
}

int b(int num){
  return num * 3;
}
```

View of memory when b() returns to a():