Runtime Stack, C Execution Model.

- How do we deal w/ nested subroutines?
- Recall: we used RT for linkage
- Can we extend this? Let's try...
- Our save area gets corrupted! Why?
- Need a "Moving Save Area". Requirements?
  - Variable size areas
    - Maintains linkage (control flow) 
    - Maintains linkage (data)
  - LIPO

Stack will store:
  - Local vars (closely tied to scope)
  - Arguments
  - Return addr
  - Linkage to prev. entry
  - An "entry" called a [stack frame / act. record]

What is it? Just memory! Every program gets one
  dynamically sized
  comes from the OS
  by convention, grows down in memory

Example: recursion
Dynamic Memory

- How do we allocate mem. for things for which the size is unknown at compile time?

- Need way to "get" new memory at any time.

- In C, we use malloc. This is a function provided by glibc (C std. library).

- Malloc gets mem. from the heap, another large block of mem. handed to us by the OS.

- Glibc implements an allocator that carves off chunks of that memory.

- Returns pointer (not size!). We must manually free the memory when we're done. Otherwise -> mem. leaks.

- Example program
Remember! It's all just memory!

- weordon off parts of it for convenience

Diagram of memory layout:
- stack
- heap
- .bss
- .data (data)
- .text (code)
- uninitialized global data
- global data (static)
- automatic mem.
- dtb memory
- program code
How does a program run? (This is not in book!)

- Intro: The kernel (user/kern sep.)
- The init process (1st userspace prog.)
- The shell... what is it?
- How does shell launch programs?
  --> syscalls! (traps)
- What is "main"? (Not actually the first thing to run)
- Notion of a runtime system: "Hidden code that helps us out..."
- Every C binary has a _start symbol
  - Kernel sets PC to here
- What about arguments? (argv, argc)