CS 61 - Lecture 05 - September 16, 2014

Grading submission:

To get a repository URL, go to code.seas.harvard.edu
Get the URL from the clone & push URLs place
Make repository private

Dynamic Memory Allocation

Storage durations

- static - (global)
- automatic - (local)
    - compiler takes care of deciding how much space to allocate per function
        - data dependent, managed automatically

    Aside: Fibonacci Recursion

    `fib(i-1)+fib(i-2)` calculates the first one first

    - only one function is called at a time
    - Example: `fib(100)` has about at most 100 functions on the stack frame

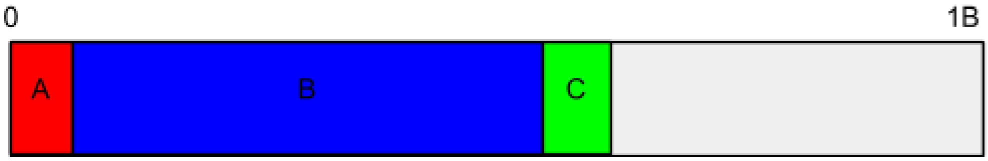
    Eager Evaluation - arguments evaluated before the function is called

    - in C arguments are called in an undetermined order
- dynamic - (exists as long as user wants it to)
    - persistent throughout entire program, outlasts functions

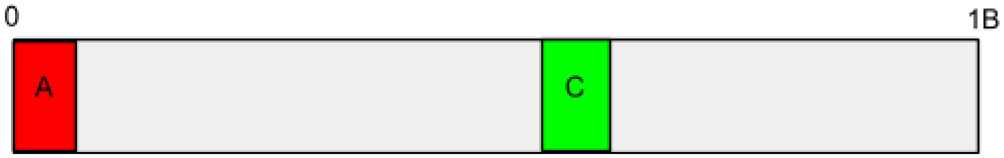
Overheads of allocators:

- Overhead - metadata
- Internal fragmentation - space inside an allocation thats not usable
    - ex: space used for alignment

    ```
    for(int i = 0; i != 1000000000; ++i)
          a[i] = malloc(1);
    ```

    On a machine with 8 byte alignment,
    this code wastes at least 7 billion bytes after allocating 1 billion bytes
- External fragmentation - space between allocations, unusable for a particular allocation
    - `a = malloc(8); b = malloc(500 000 000); c = malloc(8);`
        - 

    - `free(b);`

        - 

    - `d = malloc(700 000 000);`
        - this fails due to a lack of contiguous space
        - A and C split the open space into two smaller blocks
        - The open space is not usable for the new allocation
        - Solution: multiple heaps for different size objects

<u>Alignments:</u>

Malloc always returns something that is aligned for the maximum alignment on the machine, regardless of how much is malloc'd

`malloc(1)` is still aligned to 8 bytes on a 32-bit machine and 16 bytes on a 64-bit machine regardless of it only allocating one byte

The alignment of a struct is the LCM of all of the component alignments, but since those are all powers of 2 it is just the max

Disk fragmentation occurs for the same reason as memory fragmentations. It's also a dynamic allocation problem.

<u>Speed of allocations:</u>

Linux time command - run the following program, and return the time that it took to run

    ex: time ./membench-malloc

<u>Vocabulary:</u>

arena - region from which we allocate memory

free list - linked list of free space