Let's Write a Best-Fit Logarithmic Allocator

Let's implement dynamic allocation!

We'll **start simple: no reclamation**.

Then we'll write
- **a best-fit logarithmic allocator**,
- which was common for a couple of decades.

 © 2018 Steven S. Lumetta. All rights reserved.

For Simplicity, We Build on Top of `malloc`

To avoid overriding the C library,
- we use `malloc` instead of `sbrk`
- to get a big chunk of memory to manage,
- and store the chunk in file-scope variables.

In particular,

```
static uint8_t* free_bytes;
static size_t n_free_bytes;
```

The **free memory consists of `n_free_bytes` bytes starting at address `free_bytes`**.

 © 2018 Steven S. Lumetta. All rights reserved.

First Step: Carving Off a Block

How do we allocate a new block?

If we don't care about reclamation
- (reusing blocks that are freed),
- carving off a block is straightforward.

We'll write a function for doing so:

```
void* mem220_allocate
    (size_t n_bytes);
```

The behavior is identical to that of `malloc`.

 © 2018 Steven S. Lumetta. All rights reserved.

An Overly Simple Allocation Routine

```
void* mem220_allocate (size_t n_bytes)
{
    void* new_block = free_bytes;
    if (n_free_bytes < n_bytes) {
        return NULL;
    }
    free_bytes += n_bytes;
    n_free_bytes -= n_bytes;
    return new_block;
}
```

New block starts at start of free memory.

 © 2018 Steven S. Lumetta. All rights reserved.