Check Whether Available Memory is Sufficient

```
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;
}
```

Do we have enough free memory?

Remove the Block from Free Memory and Return It

```
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;
}
```

Remove the block from free memory.

And return the new block.

Should Add Alignment or Round Up Block Sizes

What about alignment?

In our **next implementation**,
- all blocks will be 2^k bytes for some integer **k**
- and the smallest will be 32 bytes (on the lab machines),
- so all blocks **will maintain `malloc`'s alignment** (typically 16-byte).

To align, round up, then squash the low bits
- X = (X + 15) & -16
- X = (X + 15) ^ ((X + 15) & 15) // safer

Want to Bin Block Sizes and Make Tracking Easy

How should we manage allocated blocks?

Without binning block sizes in some way,
- **fragmentation effects can become bad**,
- especially when coupled with alignment.
- Have you ever played "continuous Tetris?"

Allowing **arbitrary addresses** also **makes tracking blocks more difficult** (and pointers have alignment requirements, too).

