Final Routine is Identical to `free`

The last routine behaves identically to **`free`**:

```
void mem220_free (void* ptr);
```

Note that blocks from **C**'s library API are not interchangeable with blocks from our API.

Blocks allocated with our routines must be freed with `mem220_free`.

When Does Non-Trivial Initialization Occur?

Remember **our file-scope variables**?

```
static uint8_t* free_bytes;
static size_t n_free_bytes;
static mem_block_t*
    mem_bin[MEM220_MAX_ALLOC_LOG+1];
```

You may have noticed that they **are not initialized**.

When does initialization take place?

And how do we cause it to happen?

When Can Non-Trivial Initialization Occur?

What are the options?

1. **static initialization** (`static int x = 42;`) — not a solution for our problem
2. **a new API call** (`int32_t mem220_init (void);`) — requires that other code call it first
3. **compiler/language/Makefile support** (available in **C++**) — only last available in **C**, and not always easy to use anyway
4. **on first API call** (check in every call) — requires extra work for every call

Which API Calls Can Be Made First?

Which can the user call first?

`mem220_allocate`

`mem220_allocate_and_zero`

`mem220_reallocate`

~~`mem220_free`~~

But

- **`mem220_allocate_and_zero` and `mem220_reallocate`**
- **call `mem220_allocate`!**

So only one call need be checked…

