https://youtu.be/7fBkcIL6d8k

Static Members

Class members can be declared `static`, which means that the member belongs to the entire class, instead of to a specific instance of the class. More specifically, a `static` member is created only once and then shared by all instances (i.e. objects) of the class. That means that if the `static` member gets changed, either by a user of the class or within a member function of the class itself, then all members of the class will see that change the next time they access the `static` member.

QUIZ QUESTION

Imagine you have a `class Sphere` with a `static int counter` member. `Sphere` increments `counter` in the constructor and uses this to track how many `Sphere`s have been created. What would happen if you instantiated a new classes (`Cube`, for instance) that also had a `static int counter`? Would the two `counter`s conflict?

- Yes, instantiating a class of a different name that has a static attribute `counter` will increment the same `counter` as before.
- ✓ No, because the new static attribute `counter` is defined within the `Cube` class, it has nothing to do with `Sphere::counter`.
- Only if both classes are instantiated within the same scope do the two `counter` attributes conflict.

SUBMIT

Implementation

`static` members are **declared** within their `class` (often in a header file) but in most cases they must be **defined** within the global scope. That's because memory is allocated for `static` variables immediately when the program begins, at the same time any global variables are initialized.

Here is an example:

```
#include <cassert>

class Foo {
 public:
  static int count;
  Foo() { Foo::count += 1; }
};

int Foo::count{0};

int main() {
  Foo f{};
  assert(Foo::count == 1);
}
```

An exception to the global definition of `static` members is if such members can be marked as `constexpr`. In that case, the `static` member variable can be both declared and defined within the `class` definition:

```
struct Kilometer {
  static constexpr int meters{1000};
};
```

Exercise: Pi

`class Sphere` has a member `const double pi`. Experiment with specifying `pi` to be `const`, `constexpr`, and `static`. Which specifications work and which break? Do you understand why?

Saving Graffiti Recording. Please wait...