Class Fields Look Exactly Like `struct` Fields

Fields are **exactly the same as** in a `struct`.

Add lines that **look like variable declarations**

```
int32_t   x;
double    y;
player_t* p;
```

and each instance has fields named `x`, `y`, and `p`.

Order in memory is order of declaration.

Fields come after fields from parent class (as with best practice for subtypes in C).

Class Variables Prefixed with `static`

To **declare** the existence of a **static variable**
- called a **class variable**—one for the class, not one per instance,
- **prefix** the declaration **in the class definition with `static`**:

```
static int32_t   classInt;
static double    classDouble;
static player_t* classPlayer;
```

These variables **reside in** the **global data** area.

Declaration in Class Definition is Not Sufficient

```
static int32_t classInt;
```

The **line above**
- declares the existence of `classInt`,
- but **does not create storage**,
- so **`classInt` cannot be initialized in the class definition**.

Static variables must also be declared outside of the class definition.

Declare Class Variables in a Source File as Well

```
static int32_t classInt;
```

Outside of `MyClass`' definition, **the variable above is called `MyClass::classInt`**.[+]

It **must be declared exactly once**
- **outside of the class definition**
- (so not in a header file)
- **and can be initialized there**.

[+] :: is a new operator.

