Common to Pass a Pointer to a Struct to Functions

Think back to some of the functions that we defined for use with structures…

For stacks, we defined **`stack_init`**, **`stack_empty`**, **`stack_full`**, **`stack_push`**, and **`stack_pop`**.

For players, we defined **`player_init`**, **`player_start_game`**, **`player_finish_game`**, and **`player_delete`**.

All of them took a pointer to the associated `struct`!

Member Functions Have an Implicit Class* Argument

C++ redefines syntax to make declaration of functions associated with a class easier.

A line such as

```
int32_t memFunc (char x, double* y);
```

declares a **member function** or **method**.

Member functions
- **have an implicit first argument**,
- a pointer to an instance of the class.

With the implicit first argument, **member function implementations obey the usual calling convention**.

Member Functions' Implicit Argument is Called `this`

In other words, if the function

```
int32_t memFunc (char x, double* y);
```

appears in the definition **`MyClass`**, the **actual function signature**, in **C** syntax, **is**

```
int32_t memFunc (MyClass* this,
                 char x, double* y);
```

Notice that
- **`MyClass*` argument is always first**, and
- **the name, `this`, is** also **implicit**.

Member Functions Used as If They Were Fields

Member functions are **used as if they were fields**. Given

```
MyClass a;

MyClass* ptr;
```

we can **call `memFunc` by writing**

```
a.memFunc ('z', NULL); // this is &a

ptr->memFunc ('z', NULL);
                      // this is ptr
```

