CPP-07

* Using Pointers for class

⇒ Pointer can point to objects of Custom classes.

⇒ Call object functions from pointer with →.

obj → Func() ⇔ (*obj).Func()

* Pointers are polymorphic

⇒ Pointers are just like references, but have additional useful properties:

- → Can be reassigned
- → Can point to nothing
- → Can be stored in a Vector or an array.

* Use of pointer polymorphism

Derived derived;
Base* ptr = &derived.

* this pointer

⇒ Every object of a class or struct holds a Pointer to itself.

⇒ This pointer is called this.

⇒ Allows the object to

- # Return pointer to themselves (return * this)
- # Create a copy of themselves within a function
- # Explicitly show that a member belongs to

the current object: this->x();

* Using const with pointers

⇒ Pointers can point to a const variable:
  ↳ Cannot change value, can reassign pointer.

```
Const MyType* var_ptr = &var;
```

⇒ Pointer can be const.
  ↳ Cannot reassign pointer, can change value.

```
MyType* Const var_ptr = &var;
```

⇒ Pointer can do both at the same time:
  ↳ Cannot change in any way, readonly.

```
Const MyType* const var_ptr = &var;
```

* Memory management Structure

⇒ Working Memory is divided into two parts:
  ↳ Stack
  ↳ Heap

* Stack memory

→ Static memory
→ Small / limited
→ Memory allocation is fast
→ LIFO (Last in First out) structure.
→ Push = add
→ POP = remove.

* Heap memory

⇒ Dynamic memory
⇒ Available for long time
⇒ Raw modification possible with new and delete.
⇒ Allocation is slower than stack allocations.

* Operators new and new[]

⇒ User controls memory allocation (unsafe)
⇒ Use of new to allocate data:

```
int* int_ptr = nullptr;
int_ptr = new int;
```

* Operators delete and delete[]

⇒ Memory is not freed automatically.
⇒ Use delete to free memory

```
delete int_ptr;
```

* Memory leak

↳ memory allocated on Heap access to which has been lost.

* RAII

↳ Resource Allocation is Initialization.

* Shallow vs deep copy

Just copy pointer, not data

Copy data, Create new pointer

⇒ Default copy constructor and assignment operator implement shallow copying.

* Smart pointer

⇒ Smart pointer wrap a raw pointer into a class and manage its lifetime (RAII)

⇒ Smart pointers are all about ownership.

⇒ Always use smart pointers when the pointer should own heap memory.

⇒ Only use them with heap memory.

⇒ Still use raw pointer for non-owning pointers & simple address storing.

⇒ #include <memory> to use smart pointer

⇒ We will focus on 2 type of smart pointer.

- std::unique_ptr
- std::shared_ptr