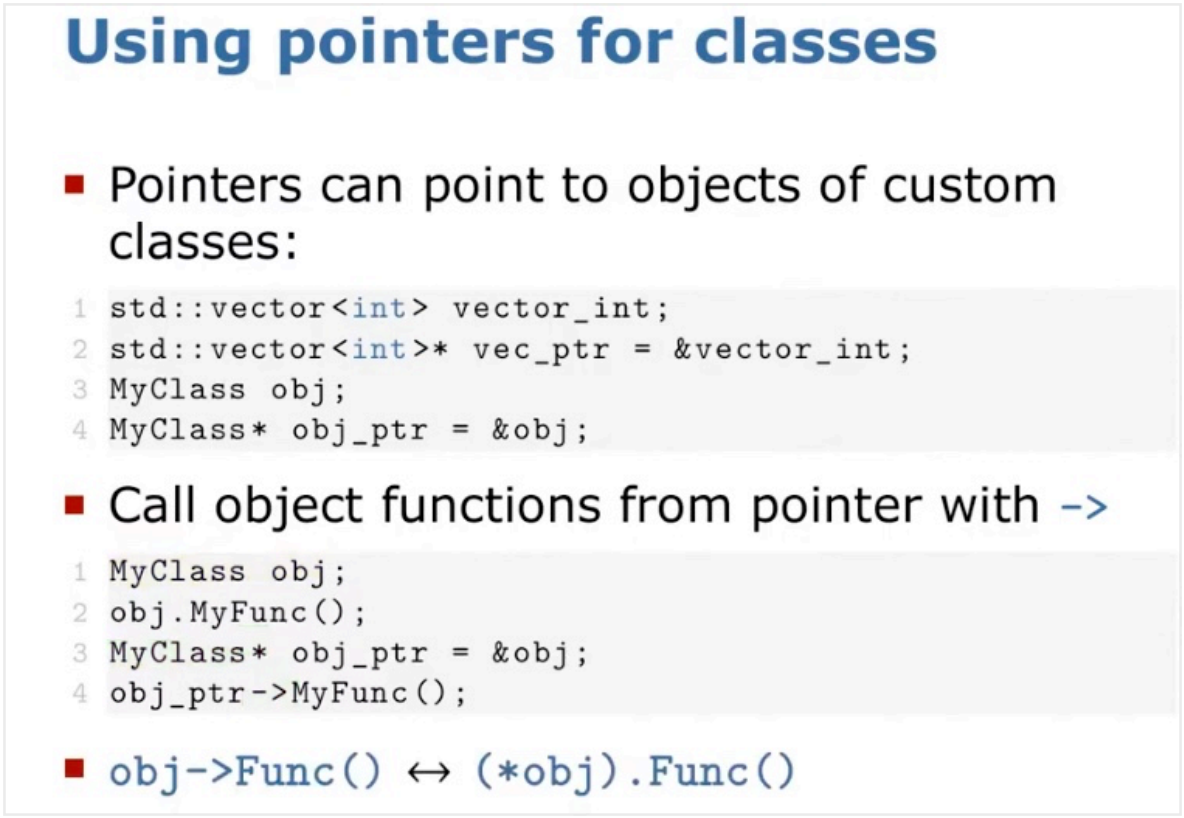
Pointer to Classes



Pointers are polymorphic

- Pointers are just like references, but have additional useful properties:
  - Can be reassigned
  - Can point to “nothing” (nullptr)
  - Can be stored in a vector or an array
- Use pointers for polymorphism
  Derived derived;
  Base* ptr = &derived;

```cpp
struct AbstractShape {
  virtual void Print() const = 0;
};
struct Square : public AbstractShape {
  void Print() const override { cout << "Square\n"; }
};
struct Triangle : public AbstractShape {
  void Print() const override { cout << "Triangle\n"; }
};

int main() {
  std::vector<AbstractShape*> shapes;
  Square square;
  Triangle triangle;
  shapes.push_back(&square);
  shapes.push_back(&triangle);
  for (const auto& shape : shapes) {
    shape->Print();
  }
  return 0;
}
```

this pointer

- Every object of a class or a struct holds a pointer to itself
- This pointer is called **this**
- Allows the objects to:
  - Return a reference to themselves: return *this;
  - Create copies 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:

```
1 // Cannot change value, can reassign pointer.
2 const MyType* const_var_ptr = &var;
3 const_var_ptr = &var_other;
```

- Pointers can **be** `const`:

```
1 // Cannot reassign pointer, can change value.
2 MyType* const var_const_ptr = &var;
3 var_const_ptr->a = 10;
```

- Pointers can do both at the same time:

```
1 // Cannot change in any way, read-only.
2 const MyType* const const_var_const_ptr = &var;
```

- Read from right to left to see which const refers to what