



CIS 3990 Recitation 02

Objects

2025-09-11

Agenda

01 Logistics

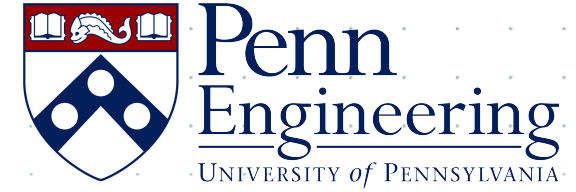
02 Objects

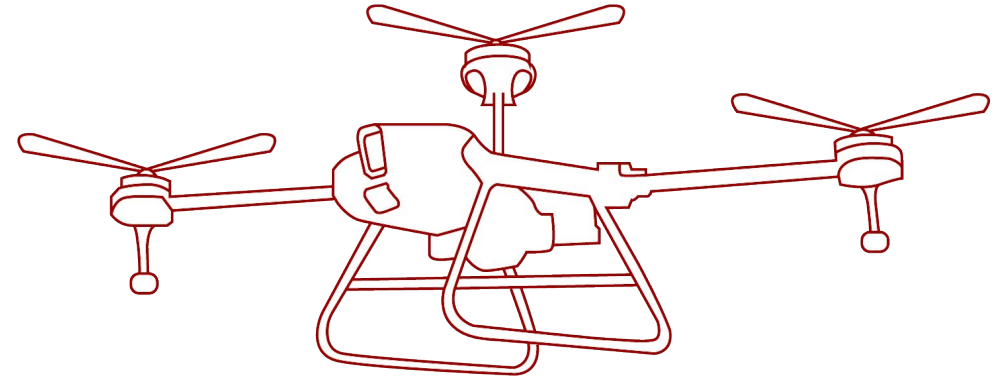
03 Operators

04 STL

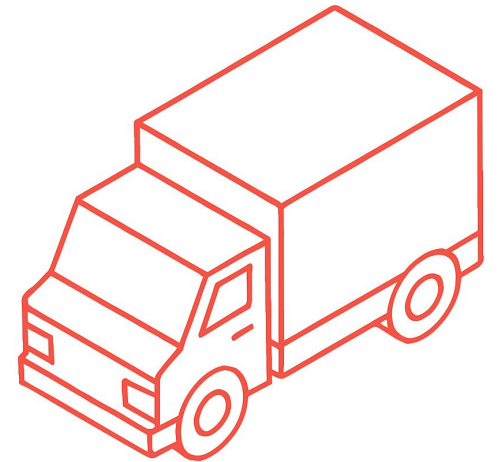
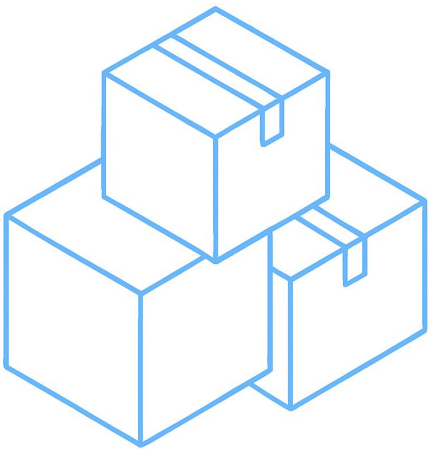
05 FSMs

06 Ed practice





Logistics



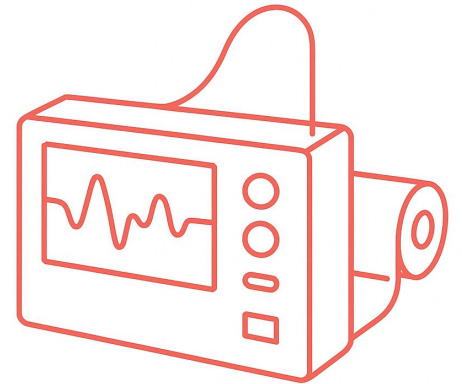
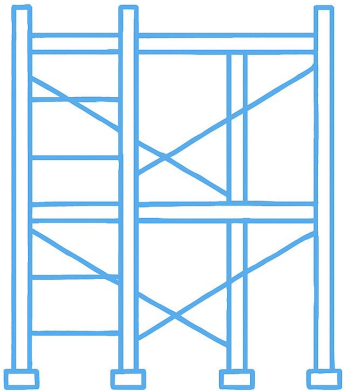
Logistics

- HW00 and HW01 past-due - but can be reopened via the Check-Ins
- Survey00 out - due 11:59pm on Fri, Sep 12 - via Canvas
- Check-in 01 out - due before lecture on Mon, Sep 15 - via Ed
- HW02 out - due 11:59pm on Tue, Sept 16
- Exam 0 - Oct 22 - Please let us know ASAP if you have conflicts
- Ash's OH - currently Mon 4:45pm-5:45pm & Fri 12:15pm-1:15pm
- When would you benefit from OH?
- Theodor's OH - still waiting on confirmation from Admin for new time



Classes

i.e. fancy structs



What classes add

Unlike C structs, C++ classes have **methods**, **access modifiers**, and **inheritance**

- **Methods** are functions that are members of the class
- **Access modifiers** affect where the methods/fields inside the class can be accessed (public vs private)
- **Inheritance** allows classes to derive (inherit) properties from existing classes

Constructors are used to instantiate a new object instance with the class name as the method name

- Can have multiple with different parameters

Constructors

What gets called when we want a new object

Constructors

What gets called when we want a new object



Constructors

What gets called when we want a new object

- Default - no arguments

Constructors

What gets called when we want a new object

- Default - no arguments
- Parametrized - some arguments, custom behavior

Constructors

What gets called when we want a new object

- Default - no arguments
- Parametrized - some arguments, custom behavior
- Copy - reference to another object, deep copy

Constructors

What gets called when we want a new object

- Default - no arguments
- Parametrized - some arguments, custom behavior
- Copy - reference to another object, deep copy
- Move - reference to another object, deep move

Constructors example

- Define the class Str as follows:

```
class Str {  
    private:  
        char* data;  
        size_t capacity;
```

Constructors example

- Default constructor

```
class Str {  
private:  
    char* data;  
    size_t capacity;  
  
public:  
    // Default constructor  
    Str() {  
        data = new char[11];  
        memset(data, '\0', 11);  
        capacity = 10;  
    }  
}
```

Constructors example

- Parameterized constructor

```
// Parameterized constructor
Str(const char* str) {
    data = new char[strlen(str) + 1];
    capacity = strlen(str);
    strcpy(data, str);
}
```

Constructors example

- Copy constructor

```
// Copy constructor
Str(const Str& other) {
    data = new char[other.capacity + 1];
    strcpy(data, other.data); // deep copy
    capacity = other.capacity;
}
```

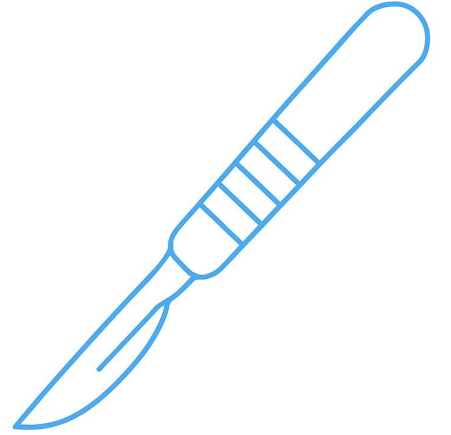
Constructors example

- Move constructor

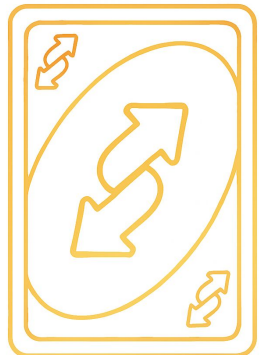
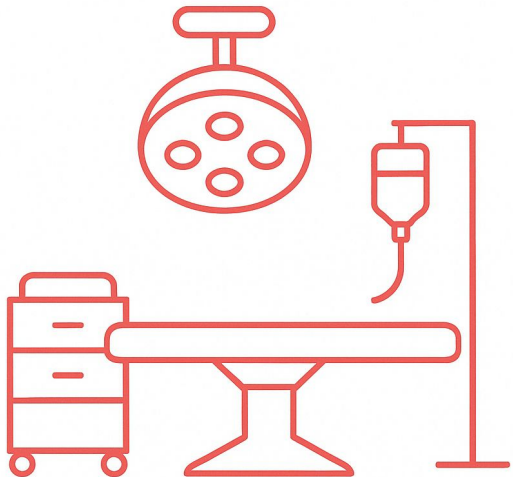
```
// Move constructor  
Str(Str&& other) noexcept {  
    data = other.data;  
    capacity = other.capacity;  
    other.data = nullptr;  
}
```

Live demo





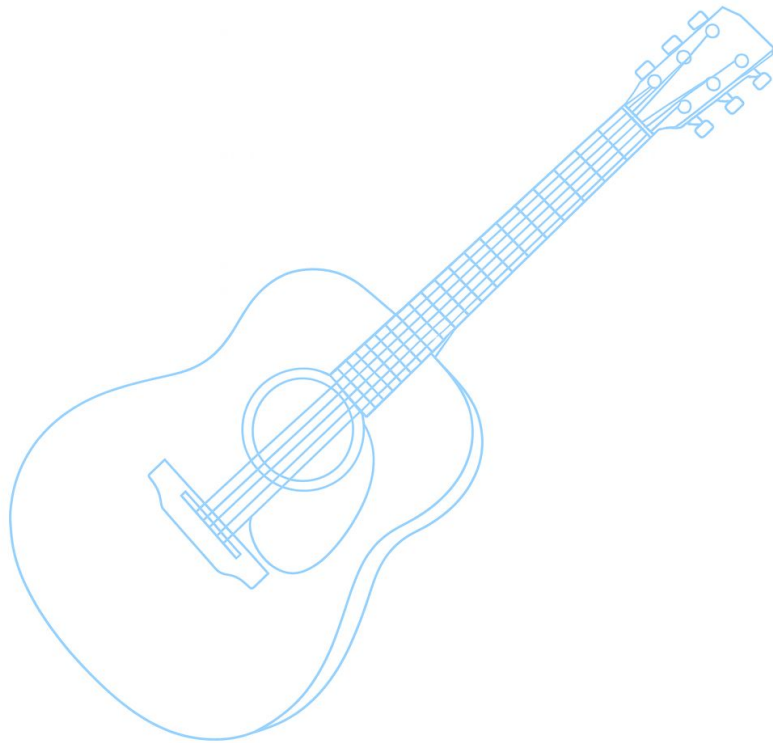
Operators



You can overload operators

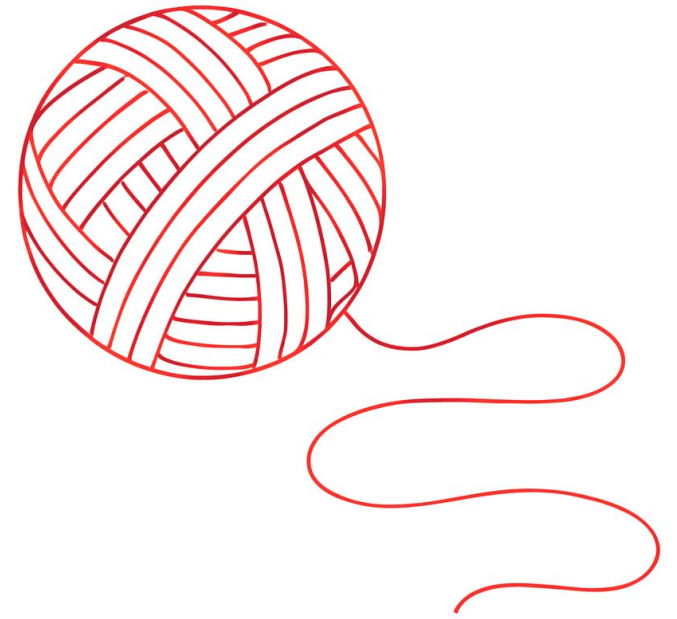
- You already saw it with the << operator for output streams in the demo
- Rule of 5:
 - Destructor: ~ClassName()
 - Copy Constructor: ClassName(const ClassName&)
 - **Copy Assignment Operator: ClassName& operator=(const ClassName&)**
 - Move Constructor: ClassName(ClassName&&)
 - **Move Assignment Operator: ClassName& operator=(ClassName&&)**
- Also useful to define comparison, addition, etc IF THEY MAKE SENSE FOR YOUR CLASS

Let's add them



STL

Cpp's peace offering



Standard Template Library

Cpp adds **a lot** of syntax and concepts compared to C, while also supporting almost all C code, making it feel clunky to use.

STL hides away the pain and complexity for commonly-used containers!

```
00369 #ifdef __GXX_EXPERIMENTAL_CXX0X__
00370     template<typename _Up = _Tp>
00371         typename __gnu_cxx::__enable_if<!std::__are_same<_Up, bool>::__value,
00372             void>::__type
00373         push_back(_Tp&& __x)
00374     { emplace_back(std::move(__x)); }
00375
00376     template<typename... _Args>
00377     void
00378     emplace_back(_Args&&... __args)
00379     {
00380     bool __realloc = _M_requires_reallocation(this->size() + 1);
00381     _Base::emplace_back(std::forward<_Args>(__args)...);
00382     if (__realloc)
00383         this->_M_invalidate_all();
00384     _M_update_guaranteed_capacity();
00385     }
00386 #endif
00387
```



```
vector<Str> my_vec;
my_vec.push_back("Abc");
```

Theo's favorites

- vector - for "arrayLists"
- algorithm - for sorting
- pair - for grouping values without defining structs
- deque - both stack and queue
- set, map - ordered by key
- unordered_set, unordered_map - hash-based

Sort

- Works on vectors of any object that defines the '<' operator, or any objects for which you provide a compare function
- Also works on C-style arrays
- Requires start and end iterators/ pointers
- Demo now

std::sort

Defined in header `<algorithm>`

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );
```

(1) (constexpr since C++20)

```
template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy,
           RandomIt first, RandomIt last );
```

(2) (since C++17)

```
template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );
```

(3) (constexpr since C++20)

```
template< class ExecutionPolicy, class RandomIt, class Compare >
void sort( ExecutionPolicy&& policy,
           RandomIt first, RandomIt last, Compare comp );
```

(4) (since C++17)

Pair

- Combines two types into a new type
- Automatically defines comparison, assignment, etc.
- Great for quick and dirty code
- Demo with `vector<pair<Str, int>`

chalk & talk

Ed practice

CIS 3990 – Ed Lessons



Lessons

 Search lessons

 Refresh

 New Module

 Import Lesson

 New Lesson

Lecture 00



• Pointers and Memory



 C Programming Refresher





That's all Folks!