Today’s Agenda

- Assn2
- Cool new features in C++17, C++20
- Future directions in CS
Assignment 2
Milestone 2

- Due tonight!
- Let us know if you need extra time :)
void global_func(const Obj& a, const Obj& b) {
    a.foo();
    b.foo();
}

Obj::foo() const {
    // needs to be const, or compilation error
    ...
}
Indexing ([])  

If key exists, returns reference to mapped value. Otherwise, returns default value. 

Hint: what does the `insert` function do?

```cpp
template <typename K, typename M, typename H>
std::pair<typename HashMap<K, M, H>::iterator, bool> HashMap<K, M, H>::insert(const value_type& value) {
    const auto& [key, mapped] = value;
    auto [prev, node_to_edit] = find_node(key);
    size_t index = _hash_function(key) % bucket_count();

    if (node_to_edit != nullptr) {
        return {make_iterator(node_to_edit), false};
    }

    auto temp = new node(value, _buckets_array[index]);
    _buckets_array[index] = temp;
    ++_size;
    return {make_iterator(temp), true};
}
```
Stream Insertion (<<)

If you have the elements 1, 3, 5, 7, 9:
need to generate “{1, 3, 5, 7, 9}”  (note: no comma on last element!)

How to do it? Try **stringstreams** (detailed in Lecture 4). There are also alternative solutions.

```cpp
std::ostringstream oss("a");
oss << "bcdef";
std::string out = oss.str();  // “abcdef”
```
Equality (==)

How to check map equality?
Hint: you only need to loop through one HashMap.
Move Constructor

Reminder: moving a vector

```
vector<T>(vector<T>&& other):  
    _size(std::move(other._size)),  
    _capacity(std::move(other._capacity)) {
    // steal the other array  🕵️
    _elems = std::move(other._elems);

    other._elems = nullptr;
    other._size = 0;
}
```

Just set everything equal to `std::move(other.attr)`. 
Move Assignment

Very similar, except we set everything = instead of initializer list

```cpp
vector<T>::operator=(vector<T>&& other) {
    _size = std::move(other._size),
    _capacity = std::move(other._capacity),
    _elems = std::move(other._elems)
// steal the other array 🕵
    other._elems = nullptr;
    other._size = 0;
}
```
Variadic templates
(C++11)
Variadic templates

Allow for templates with a **variable** number of arguments!

```cpp
template< typename T >
T adder(T v) {
    return v;
}

template< typename T, typename ... Args >
T adder(T first, Args... args) {
    return first + adder(args...);
}

dadder(5, 6, 7, 8) // 26
```
template<
typename T>
T adder(T v) {
    return v;
}

template<
typename T,
typename... Args>
T adder(T first, Args... args) {
    return first + adder(args...);
}

How does it work?
Overload resolution!
Spaceship operator
(C++17)
Writing this boilerplate code is annoying:

```cpp
struct IntWrapper {
    int value;
    IntWrapper(int value): value{value} { }
    bool operator==(const IntWrapper& rhs) const { return value == rhs.value; }
    bool operator!=(const IntWrapper& rhs) const { return !(*this == rhs); }
    bool operator<(const IntWrapper& rhs) const { return value < rhs.value; }
    bool operator<=(const IntWrapper& rhs) const { return !rhs < *this; }
    bool operator>(const IntWrapper& rhs) const { return rhs < *this; }
    bool operator>=(const IntWrapper& rhs) const { return !(*this < rhs); }
};
```
If you write a single $\leftrightarrow$ operator, everything will be autogenerated for you.

```cpp
struct IntWrapper {
    int value;
    IntWrapper(int value): value(value) { }
    auto operator<=>(const int& rhs) auto {
        return value <=> rhs;
    }
};

IntWrapper(5) < IntWrapper(7)    // returns true
```
Spaceship operator

Basically, return -1, 0, or 1 as appropriate:

```cpp
struct IntWrapper {
    int value;
    IntWrapper(int value): value(value) { }
    auto operator<=>(const int& rhs) auto {
        if (value < rhs) return -1;
        else if (value == rhs) return 0;
        else return 1;
    }
};

IntWrapper(5) < IntWrapper(7)    // returns true
```
Designated initializers

(C++20)
Better struct initialization syntax!

Non-specified values → default initialization

```c
struct A {
    int x;
    int y;
    int z = 123;
};

A a {.x = 1, .z = 2};  // a.x == 1, a.y == 0, a.z == 2
```
[[likely]]
(C++20)
int random = get_random_number_between_x_and_y(0, 100);
[[likely]] if (random > 0) {
    // body of if statement; efficiency will be prioritized
}

[[unlikely]] if (random == 0) {
    // body of if statement; efficiency will not be prioritized
}
How does this work?

The Meltdown and Spectre exploits use “speculative execution?” What’s that?

You know the Trolley problem? Well, for a while now, CPUs have basically been sending trolleys down both paths, quantum-style, while awaiting your choice. Then the unneeded “phantom” trolley disappears.

The Phantom trolley isn’t supposed to touch anyone. But it turns out you can still use it to do stuff, and it can drive through walls.
Future Directions in CS
AI

Coding

CS 106A

CS 106B

CS 107

CS 110

CS 109

CS 103

CS 161

Theory/math

Machine Learning (mathy)

CS 229

CS 124

CS 231N

Deep Learning for Vision

CS 224W

ML on Graphs

Reinforcement Learning

CS 234

Deep NLP

CS 224N

CS 223A

Robots
Advanced courses:
- CS 229M: Statistical machine learning theory
- CS 329D: ML under distributional shift
- CS 468: ML in non-euclidean spaces
Biocomputation

CS 106A

CS 106B

CS 107

CS 110

CS 109

CS 161

CS 103

CS 274

Representations and Algorithms for Computational Molecular Biology

CS 279

Biomolecules

CS 275

Translational Informatics

Coding

Theory/math
Other fields

- **Theory**: theory of computation, crypto, algorithm design
- **HCI**: interface design, Going Viral
Interdisciplinary

- **CS 209**: Law, Bias, and Algorithms (with LAW)
  - (what can we do to make our systems more fair and transparent?)
- **CS 275B**: Computational Music Analysis (with MUSIC)
- **CS 342**: Building for Digital Health (with MED)
Going Forward from CS106L

CS110: Principles of Computer Systems
Winter 2021
Mo/We/Fr 1pm-2:20pm PDT via Zoom (link on Canvas)

(Uses a *lot* of C++.)
Going Forward from CS106L

CS 110L: Safety in Systems Programming

(Covers Rust—a powerful language which offers C++ performance, without the unsafe stuff. In Spring!)
Going Forward from CS106L

CS 41: The Python Programming Language

Tuesday & Thursday @ 2:30pm - 3:50pm
Join URL: See you in Spring!
Contribute to our Spotify Playlist!

(Python—a powerful language that needs no introduction. Taught in Spring, or sign up to be a TA)
Going Forward from CS106L

Get a job with your C++ skills!

<table>
<thead>
<tr>
<th>12/11/2020</th>
<th>Tower Research Capital</th>
<th>Full Time: Experienced Software Developer, C++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Position Details</td>
</tr>
</tbody>
</table>

Firefox C++ Development Intern

Mozilla is hiring C++ Software Engineering Interns onto our technical teams throughout the world. Our headquarters are based in the Bay Area, but these two opportunities are located at our office in Paris!

FACEBOOK Careers

Software Engineer, Intern/Co-op Responsibilities

- Code high-volume software using primarily C++ and Java
Research: CURIS

- Research projects in all areas of CS!
- Projects are available all year; applications close soon

<table>
<thead>
<tr>
<th>Foundation of Algorithmic Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor: Omer Reingold</td>
</tr>
<tr>
<td>Fields: Theory of Computation</td>
</tr>
<tr>
<td>Quarter: Win_spr</td>
</tr>
<tr>
<td>Compensation: Paid or credit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tock: Secure Embedded Operating Systems Design in Rust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor: Philip A Levis</td>
</tr>
<tr>
<td>Fields: Operating Systems, Security</td>
</tr>
<tr>
<td>Quarter: Aut_win_spr</td>
</tr>
<tr>
<td>Compensation: Paid or credit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Medical imaging AI in COVID-19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor: Daniel Rubin</td>
</tr>
<tr>
<td>Fields: AI, Vision, Algorithms</td>
</tr>
<tr>
<td>Quarter: Aut_win</td>
</tr>
<tr>
<td>Compensation: Course credit</td>
</tr>
</tbody>
</table>
Thank you

for all of your support