C++ STL (part 1 of 2)

CSE 333 Fall 2023

Instructor: Chris Thachuk

Teaching Assistants:

Ann Baturytski
Yuquan Deng
Noa Ferman
James Froelich
Hannah Jiang
Yegor Kuznetsov
Humza Lala
Alan Li
Leanna Mi Nguyen
Chanh Truong
Jennifer Xu

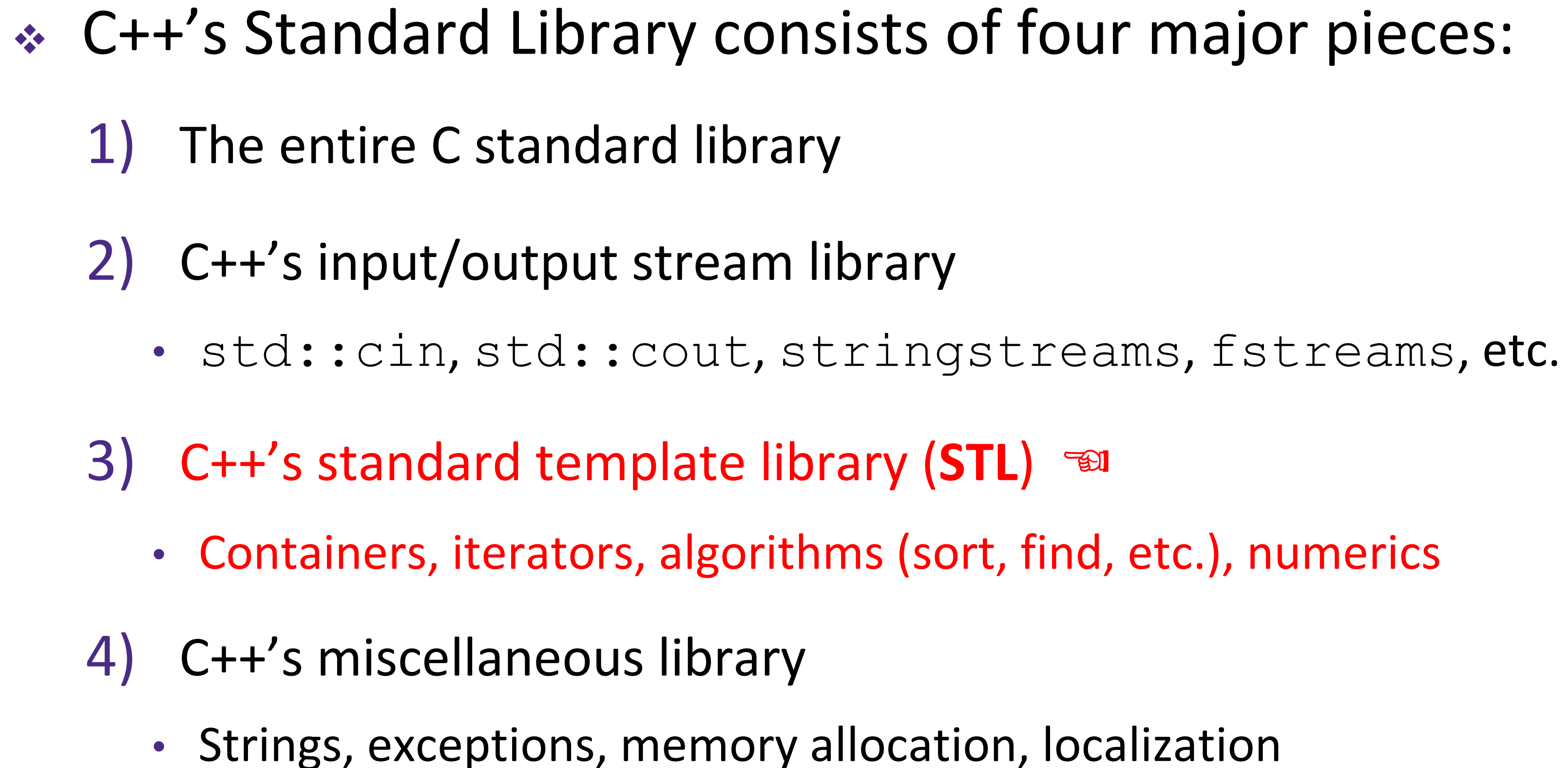
Relevant Course Information

- Exercise 7 due tonight by 10pm
- Exercise 8 due Friday by 10pm
- Homework 2 was due last night
  - Don't forget to clone your repo to double-/triple-/quadruple-check compilation!
- Homework 3 will be released by tomorrow, due 11/22

Lecture Outline

- **STL overview**
- STL iterators, algorithms *(next lecture)*

C++’s Standard Library



STL Containers ☺

- A container is an object that stores (in memory) a collection of other objects (elements)
  - Implemented as class templates, so hugely flexible
  - More info in *C++ Primer* §9.2, 11.2

- Several different classes of container
  - Sequence containers (`vector`, `deque`, `list`, `...`) index numerically
  - Associative containers (`set`, `map`, `multiset`, `multimap`, `bitset`, `...`) index by key
  - Differ in algorithmic cost and supported operations

STL Containers ☹

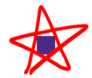
- STL containers store by *value*, not by *reference*
  - When you insert an object, the container makes a *copy*
  - If the container needs to rearrange objects, it makes copies
    - *e.g.*, if you sort a `vector`, it will make many, many copies
    - *e.g.*, if you insert into a `map`, that may trigger several copies
  - What if you don't want this (disabled copy constructor or copying is expensive)?
    - You can insert a wrapper object with a pointer to the object
      - We'll learn about these "smart pointers" soon

Our Tracer Class

sets unique id_, initial value_ is id_

- Wrapper class for an `unsigned int value_`
  - Also holds unique `unsigned int id_` (increasing from `0`)
  - Default ctor, cctor, dtor, `op=`, `op<` defined
  - `friend` function `operator<<` defined
  - Private helper method `PrintID()` to return `"(id_,value_)"` as a string
  - Class and member definitions can be found in `Tracer.h` and `Tracer.cc`
- Useful for tracing behaviors of containers
  - All methods print identifying messages
  - Unique `id_` allows you to follow individual instances

STL `vector`

- A generic, dynamically resizable array
  - https://cplusplus.com/reference/vector/vector/
  -  Elements are store in *contiguous* memory locations
    - Elements can be accessed using pointer arithmetic if you'd like
    - Random access is O(1) time ← calculate address via arithmetic, then access
  - Adding/removing from the end is cheap (amortized constant time)
  - Inserting/deleting from the middle or start is expensive (linear time) must copy all following elements

pollev.com/cse333

How many Tracer objects created?

```
#include <iostream>
#include <vector>
#include "Tracer.h"

using namespace std;

int main(int argc, char** argv) {
  Tracer a, b, c;
  vector<Tracer> vec;

  cout << "vec.push_back " << a << endl;
  vec.push_back(a);
  cout << "vec.push_back " << b << endl;
  vec.push_back(b);
  cout << "vec.push_back " << c << endl;
  vec.push_back(c);

  cout << "vec[0]" << endl << vec[0] << endl;
  cout << "vec[2]" << endl << vec[2] << endl;

  return EXIT_SUCCESS;
}
```

vector/Tracer Example

vectorfun.cc

```
#include <iostream>
#include <vector>
#include "Tracer.h"

using namespace std;

int main(int argc, char** argv) {
  Tracer a, b, c;
  vector<Tracer> vec;

  cout << "vec.push_back " << a << endl;
  vec.push_back(a);
  cout << "vec.push_back " << b << endl;
  vec.push_back(b);
  cout << "vec.push_back " << c << endl;
  vec.push_back(c);

  cout << "vec[0]" << endl << vec[0] << endl;
  cout << "vec[2]" << endl << vec[2] << endl;

  return EXIT_SUCCESS;
}
```

// most containers found in libraries of same name

// construct 3 Tracer instances

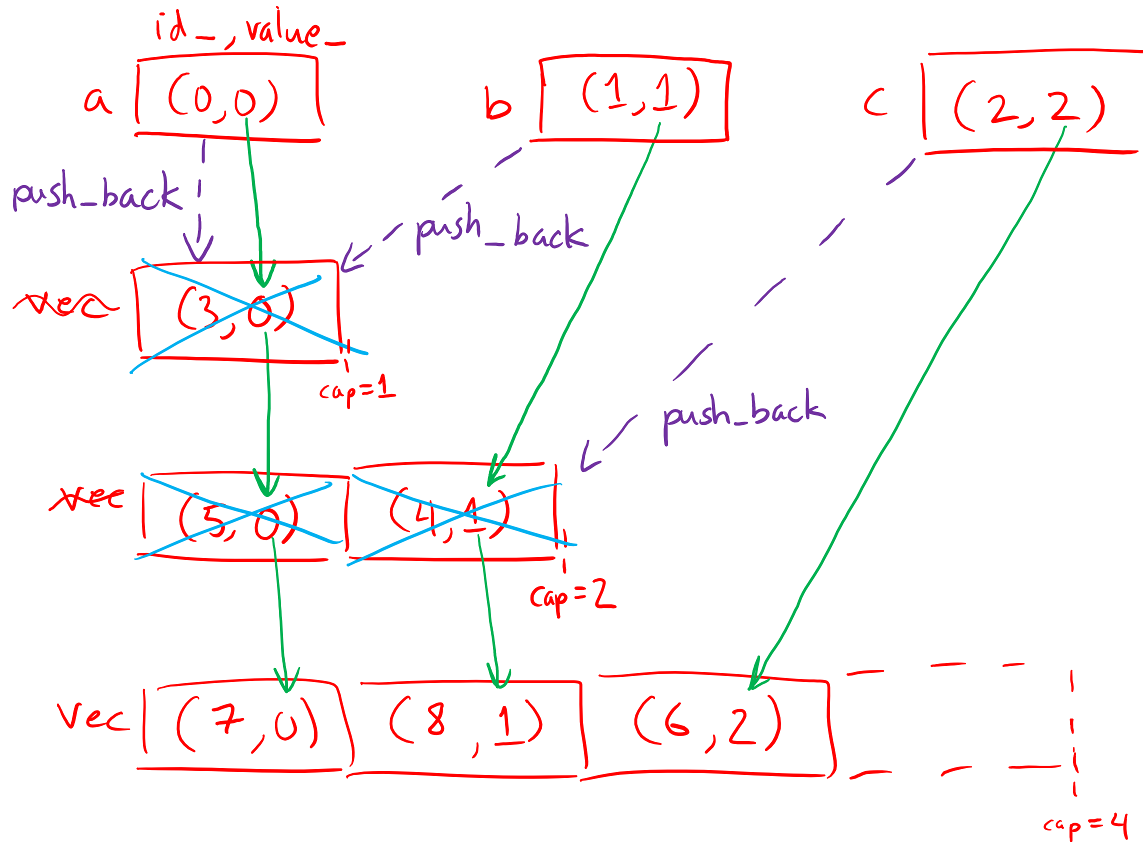
// new (empty) vector container of Tracers

add copies of Tracers to end of container

elements can be accessed via subscript notation

verify the stored values are what we expect

Why All the Copying?



- copy construction
- destruction

push_back calls	Tracers constructed
0	3 (a, b, c)
1	4
2	6
3	9
4	10
5	15

9 Tracer objects constructed!

Note: capacity doubles here each time (not an important detail)

Note: exact ordering of construction when vec gets moved not important

Extra Exercise #1

- Using the `Tracer.h/.cc` files from lecture:
  - Construct a vector of lists of Tracers
    - *i.e.*, a `vector` container with each element being a `list` of `Tracer`s
  - Observe how many copies happen ☺
    - Use the sort algorithm to sort the vector
    - Use the `list.sort()` function to sort each list

Extra Exercise #1

- Using the `Tracer.h`/`.cc` files from lecture:
  - Construct a vector of lists of Tracers
    - *i.e.*, a `vector` container with each element being a `list` of `Tracer`s
  - Observe how many copies happen ☺
    - Use the sort algorithm to sort the vector
    - Use the `list.sort()` function to sort each list