C++ STL (part 1 of 2)
CSE 333 Spring 2023

Instructor: Chris Thachuk

Teaching Assistants:

Byron Jin          CJ Reith
Deeksha Vatwani   Edward Zhang
Humza Lala         Lahari Nidadavolu
Noa Ferman         Saket Gollapudi
Seulchan (Paul) Han Timmy Yang
Tim Mandzyuk       Wui Wu
Which concept has given you the most difficulty so far in the context of Homework 2?

A. The data structures  
B. C-string manipulations  
C. POSIX I/O  
D. Dynamic memory allocation  
E. GDB  
F. Style considerations  
G. Prefer not to say
Relevant Course Information

❖ Exercise 7 due Monday

❖ Homework 2 was due last night
  ▪ Don’t forget to clone your repo to double-/triple-/quadruple-check compilation!
  ▪ Use late days if you can’t finish & polish your submission! They exist for a reason

❖ Homework 3 will be released by Monday, due in 3 weeks

❖ Midterm: May 4 – May 6 (1pm)
  ▪ Take home (Gradescope) and open notes
  ▪ Individual, but high-level discussion allowed (“Gilligan’s Island Rule”)
  ▪ No lecture next Friday (May 5)
C++’s Standard Library

- C++’s Standard Library consists of four major pieces:
  1) The entire C standard library
  2) C++’s input/output stream library
     • `std::cin`, `std::cout`, `stringstreams`, `fstreams`, etc.
  3) C++’s standard template library (STL)
     • Containers, iterators, algorithms (sort, find, etc.), numerics
  4) C++’s miscellaneous library
     • Strings, exceptions, memory allocation, localization
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, ...)
  - **Associative** containers (set, map, multiset, multimap, bitset, ...)
  - 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

- 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_)
  - 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
  - Elements are stored in contiguous memory locations
    - Elements can be accessed using pointer arithmetic if you’d like
    - Random access is $O(1)$ time
  - Adding/removing from the end is cheap (amortized constant time)
  - Inserting/deleting from the middle or start is expensive (linear time)
vector/Tracer Example

```cpp
#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;
    return EXIT_SUCCESS;
}
```
Why All the Copying?