

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

C++ Parametric Polymorphism

- ❖ C++ has the notion of **templates**
 - A function or class that accepts a **type** as a parameter
 - You define the function or class once in a type-agnostic way
 - When you invoke the function or instantiate the class, you specify (one or more) types or values as arguments to it
 - At **compile-time**, the compiler will generate the “specialized” code from your template using the types you provided
 - Your template definition is NOT runnable code
 - Code is **only** generated if you use your template

7

7

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

Solution #2 (you'll see this sometimes)

```
#ifndef COMPARE_H_
#define COMPARE_H_

template <typename T>
int comp(const T& a, const T& b);
#include "compare.cc"
#endif // COMPARE_H_
```

compare.h

```
#include <iostream>
#include "compare.h"

using namespace std;

int main(int argc, char **argv) {
    cout << comp<int>(10, 20);
    cout << endl;
    return EXIT_SUCCESS;
}
```

main.cc

```
template <typename T>
int comp(const T& a, const T& b) {
    if (a < b) return -1;
    if (b < a) return 1;
    return 0;
}
```

compare.cc

14

14

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

Poll Everywhere

pollev.com/cse33320su

- ❖ Assume we are using Solution #2 (.h includes .cc)
- ❖ Which is the *simplest* way to compile our program (a.out)?

A. g++ main.cc
 B. g++ main.cc compare.cc
 C. g++ main.cc compare.h
 D. g++ -c main.cc
 g++ -c compare.cc
 g++ main.o compare.o
 E. We're lost...

15

15

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

Pair Class Definition

Pair.h

```
#ifndef PAIR_H_
#define PAIR_H_

template <typename Thing> class Pair {
public:
    Pair() { }

    Thing get_first() const { return first_; }
    Thing get_second() const { return second_; }
    void set_first(Thing &copyme);
    void set_second(Thing &copyme);
    void Swap();

private:
    Thing first_, second_;
};

#include "Pair.cc"
```

#endif // PAIR_H_

Template parameters for class definition
 Could be objects, could be primitives
 Using solution #2

18

18

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

Pair Function Definitions

Pair.cc

```

template <typename Thing>
void Pair<Thing>::set_first(Thing &copyme) {
    first_ = copyme;
} Member of template class

template <typename Thing>
void Pair<Thing>::set_second(Thing &copyme) {
    second_ = copyme;
}

template <typename Thing>
void Pair<Thing>::Swap() {
    Thing tmp = first_;
    first_ = second_;
    second_ = tmp;
}

template <typename T>
std::ostream &operator<<(std::ostream &out, const Pair<T>& p) {
    return out << "Pair(" << p.get_first() << ", "
                  << p.get_second() << ")";
}

```

19

19

- W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020
- ## Review Questions (Classes and Templates)
- ❖ Why are only `get_first()` and `get_second()` `const`?
 - ❖ Why do the accessor methods return `Thing` and not references?
 - ❖ Why is `operator<<` not a `friend` function?
 - ❖ What happens in the default constructor when `Thing` is a class?
 - ❖ In the execution of `Swap()`, how many times are each of the following invoked (assuming `Thing` is a class)?

ctor ____ cctor ____ op= ____ dtor ____
- 22

22

W UNIVERSITY of WASHINGTON L13: C++ Templates & STL Intro CSE333, Summer 2020

STL `vector`

- ❖ A generic, dynamically resizable array.
 - <http://www.cplusplus.com/reference/stl/vector/vector/> Like a normal C array!
 - Elements are store in contiguous memory locations
 - Elements can be accessed using pointer arithmetic if you'd like
 - Random access is $O(1)$ time ← Pointer arithmetic, then acces
 - Adding/removing from the end is cheap (amortized constant time)
 - Inserting/deleting from the middle or start is expensive (linear time)

Need to shift all of the elements in the array

29

29