# CSE 333: Systems Programming

## Section 5

## Template specialization

# Templates versus generics

- C++ templates differ from Java generics in an important way—what is it?
  - Hint: If I construct LinkedList<Integer> and LinkedList<Double> in Java, how many instances of the List class have I generated? What about list<int> and list<double> in C++?
  - This is an important distinction for template specialization, as it turns out

# Template specialization

- Template specialization allows for different versions of the "same" templated class, struct, or function
- The general pattern for functions is:

```
// Original templated function
template<typename T>
T func(T arg1, T arg2) { ... }

// Specialized function for ints
template<>
int func(int arg1, int arg2) { ... }
```

- Which version of func gets invoked from a call to func(1, 2.3)? What is "T" taken to be in this case?

# Template specialization

```
// Original templated function
template<typename T>
T func(T arg1, T arg2) { ... }

// Specialized function for ints
template<>
int func(int arg1, int arg2) { ... }
```

- In the specialized version of func, we've replaced all occurrences of T with int. Note that the "typename T" is gone in the specialized version too
- The body of func can differ between the two functions, and multiple versions of the code are generated

# Template specialization

- Some examples of when template specialization of functions can be useful:
  - Defining a “compare” function
  - Defining a “min” or “max” function
  - Defining a semi-generic read or write function (see my post on the discussion board about homework 3)
  - … and many more

# Class/struct specialization

- Class and struct template specialization is similar to specialization of functions:

```
template<typename T>
class Example {
   ...
};
template<>
class Example<double> {
   ...
};
```

- Here we've declared a specialized version of the Example class for doubles

# Class/struct specialization

- The bad news is that functions and member variables are not shared ☹

```
template<typename T>
class Example {
 public:
  T foo() { ... }
};
template<>
class Example<double> {
 public:
  // No foo here unless we declare it again.
};
```

# Class/struct specialization

- Solution: Have template specializations of a class share a non-templatized base class that has templated functions
  - Let's not think about this right now…

# Class function specialization

- It's possible for non-templated classes to have templated functions (we'll see this in the exercise)
- To specialize these functions, declare the specialization outside of the class

```
class Example {
 public:
  template<typename T> T func() { ... }
};
template<>
double Example::func() { ... }
```

# Partial specialization

* Classes and structs (but not functions) also allow for partial specialization, which looks like this:

```
template<typename T1, typename T2>
class Example {
   ...
};
// Here one template argument is supplied
// but not the other
template<typename T2>
class Example<float, T2> {
   ...
};
```

# Wrapup

- That was a whirlwind tour, but hopefully it gave you more of a taste of what templates can do
- Some other template-related topics:
  - Default template parameters (see STL classes such as vector, for instance)
  - Template templates (this allows templated classes as template parameters)

# Section exercise

- ByteBuffer is an untyped storage class that provides (and requires) typed insertions and accesses
- It is useful for cases where the type of the data being stored isn't known until runtime
  - One can create a vector of ByteBuffers that store arbitrary types, for example
- The ByteBuffer's internal buffer grows dynamically as values are inserted

# Section exercise

- Demo time! (See bytebuffer_example.cc and bytebuffer_example_solution)

# Section exercise

- Your task is to write some function specializations for ByteBuffer<string>
- The content of strings should be laid out contiguously in the buffer. Use the offset fields of the class to keep track of where to insert next and where data for particular indices is located
- As usual, submit the code (bytebuffer.h) to the Dropbox and leave a comment on the Dropbox with your partner's name