Carey's Template Cheat Sheet

- To templatize a non-class function called bar:
  - Update the function header: int bar(int a) → template <typename ItemType> ItemType bar(ItemType a);
  - Replace appropriate types in the function to the new ItemType: { int a; float b; … } → {ItemType a; float b; …}
- To templatize a class called foo:
  - Put this in front of the class declaration: class foo { … }; → template <typename ItemType> class foo { … };
  - Update appropriate types in the class to the new ItemType
  - How to update internally-defined methods:
    - For normal methods, just update all types to ItemType: int bar(int a) { … } → ItemType bar(ItemType a) { … }
    - Assignment operator: foo &operator=(const foo &other) → foo<ItemType>& operator=(const foo<ItemType>& other)
    - Copy constructor: foo(const foo &other) → foo(const foo<ItemType> &other)
  - For each externally defined method:
    - For non inline methods: int foo::bar(int a) → template <typename ItemType> ItemType foo<ItemType>::bar(ItemType a)
    - For inline methods: inline int foo::bar(int a) → template <typename ItemType> inline ItemType foo<ItemType>::bar(ItemType a)
    - For copy constructors and assignment operators
    - foo &foo::operator=(const foo &other) → foo<ItemType>& foo<ItemType>::operator=(const foo<ItemType>& other)
    - foo::foo(const foo &other) → foo<ItemType>::foo(const foo<ItemType> &other)
  - If you have an internally defined struct blah in a class: class foo { … struct blah { int val; }; … };
    - Simply replace appropriate internal variables in your struct (e.g., int val;) with your ItemType (e.g., ItemType val;)
  - If an internal method in a class is trying to return an internal struct (or a pointer to an internal struct):
    - You don't need to change the function's declaration at all inside the class declaration; just update variables to your ItemType
  - If an externally-defined method in a class is trying to return an internal struct (or a pointer to an internal struct):
    - Assuming your internal structure is called "blah", update your external function bar definitions as follows:
    - blah foo::bar(…) { … } → template<typename ItemType>typename foo<ItemType>::blah foo<ItemType>::bar(…) { … }
    - blah *foo::bar(…) { … } → template<typename ItemType>typename foo<ItemType>::blah *foo<ItemType>::bar(…) { … }
- Try to pass templated items by const reference if you can (to improve performance):
  - Bad: template <typename ItemType> void foo(ItemType x)
  - Good: template <typename ItemType> void foo(const ItemType &x)