**Final Review**

**Templates**

* Function and usage
  + template <typename T>
    - The compiler will generate a copy of function's body by filling in T with the actual type used by the programmer
  + Set of all types
    - All fundamental types, Class object types, and any user define type with the operator<= and operator>= are defined. It can't worked with multiple different types inside the same parameter.

**Vectors**

* Function and usage
  + vector <type of variable> variablename;
  + Version of array list in Java
  + A data structure type of container use contiguous storage location for its elements.
  + Last element in the vector is null
  + Do not reallocate each time new element added to the container.
    - May allocate some extra storage that's greater than the storage strictly needed to contain its elements to accommodate for possible growth.
  + 3 properties:
    - The elements are ordered in strict linear sequence
    - The internal memories/array grow dynamically as more elements are added to the vector
    - Container is “templated”
* Iterator function and usage
  + front()
    - Access first element in the vector
  + back()
    - Access element before the last element in the vector
  + begin()
    - Return iterator to the first element in the vector
  + end()
    - Return iterator to the last element in the vector
  + push\_back()
    - Add element at the end of the vector
  + pop\_back()
    - Delete element before the last element in the vector
  + operator []
    - Accessing any element in the vector
    - Similar to array[] to access its element
  + for (vector <type of variable>::iterator variablename = vectorvariablename.begin(); variablename != vectorvariablename.end(); ++variablename) {

(same type of variable as vector) tempname = \*variablename;

cout << tempname << endl;

}

**Inheritance (is-a relationship)**

* A relationship between 2 classes and a mechanism to maintain same behavior, to reuse code/implementation, and to allow independent extensions of the original software via public classes and interfaces
* Actual class of object being referenced is impossible to predict at compiled-time
* Based vs Derived, Parent vs Child, and super vs sub have the same meaning.
  + Default visibility mode = private
  + Based = Parent = Super
    - Top of the food chain classes
  + Derived = Child = Sub
    - Dependent on Based/Parent/Super class
    - Child class can't access private members at allocate
    - Called constructor of parent class, which must be in the initializer.
    - Acquires properties/implementations of another class (parent class)
    - Inherit instance variables and member functions of super classes
* Public – Based Class Visibility
  + public members of based class able can be accessed by members of that base class, members of its derived class, and any members outside of that based and derived class.
* Private – Based Class Visibility
  + Public, Private, and Protected Derivation of Derived class
    - not available to inherit
  + Private members of based class can only be accessed by members of that base class
* Protected – Base Class Visibility
  + protected members of based class can only be accessed by members of that base class, and members of its derived class