Final Exam Review Topics

Functions Revisited

1. Global vs. member function. Explicit vs. implicit arguments.
2. Define global functions: top level, static inside class definition, static top level.
3. Operator extension/overloading:
   (a) Binary: \( \text{operator} \oplus (a,b) \)
       \( a.\text{operator} \oplus (b) \)
   (b) Prefix unary: \( *, -, ++, \text{new}, \ldots \) E.g., \( \text{operator*}() \)
   (c) Postfix unary: \( ++, -- \). E.g., \( \text{operator++}(\text{int}) \).
4. Functional composition and type compatibility: \( g(f(x)) \).
   (a) Conversion using constructor;
   (b) Conversion using a cast.

Polymorphism and Derivation

1. Some uses for derived classes: code reuse, modularity, generic programming and isolation, polymorphic collections.
2. Derived is a Base (and more).
3. Polymorphic pointer and slicing.
4. Polymorphic derivation and virtual functions (type tags and vtable).
5. Virtual destructors.
7. Pure virtual functions, abstract classes, and pure abstract classes: \( \text{virtual int f()=0.} \)
8. Multiple inheritance.
   (a) E.g., class A: public B, protected C \{ \ldots \} ;
   (b) Each instance of A contains an instance of B and an instance of C.
   (c) Members can be accessed using qualified names. E.g., A::B::x, A::C::x.
   (d) Diamond inheritance and virtual inheritance.
9. Simple vs. polymorphic derivation.
10. Containment as an alternative to simple derivation.

C++ Standard Library

1. Containers: vector, deque, list, map, set, multimap, multiset.
2. Common operations: creating an empty container, inserting, deleting, copying, scanning, ...
3. vector<T>
   (a) vect[5], vect.at(5).
   (b) Deriving from containers, adaptor classes.
(c) Iterators.
(d) STL algorithms: e.g., sort.

4. pair<T1, T2>
5. map<Key, Val>
6. Copying between containers:
   (a) assign(begin, end);
   (b) using constructor.

7. String class.
8. More on containers.
   (a) Linear containers.
   (b) Ordered containers.
      i. key() returns the associated key.
      ii. Comparison operators < and ==.

Name Visibility

1. Private (default), protected, and public derivations.
2. Data and function names: globally, in a class, in function parameter lists, and in code blocks.
4. Privacy attributes for a class member: hidden, private, protected, public.

<table>
<thead>
<tr>
<th>Attribute in base class</th>
<th>public</th>
<th>protected</th>
<th>private</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>public</td>
<td>protected</td>
<td>private</td>
</tr>
<tr>
<td>protected</td>
<td>protected</td>
<td>protected</td>
<td>private</td>
</tr>
<tr>
<td>private</td>
<td>hidden</td>
<td>hidden</td>
<td>hidden</td>
</tr>
<tr>
<td>hidden</td>
<td>hidden</td>
<td>hidden</td>
<td>hidden</td>
</tr>
</tbody>
</table>

Attribute in derived class

Template

1. Template overview.
2. Template functions and specialization:
   template <class X> void swapargs(X& a, X& b) { ... };
3. Template classes: template <class T> class FlexArray { ... };
4. Non-inline template functions must be compiled and linked for each instantiation.
   (a) Put all template function definitions in the .hpp file along with the class definition;
   (b) Or put template function definitions in the .cpp file and explicitly instantiate.
      E.g., template class FlexArray<int>;
5. Template parameters can have default values:
   template<class T=int, int size=100> class A { ... };

Casts and Conversions

1. C casts: value casts vs. pointer casts.
2. C++ casts: static casts, dynamic casts, reinterpret casts, and const casts.
3. Explicit cast syntax: C-style (double)p; function notation double(x); and cast notation static_cast<double>(x);
   Assignment: int x; double d; x=d;
4. Implicit cast:
   Pointer assignment: Base* bp; Derived* dp; bp = dp;
   Initialization: A a=x;
   Construction: A a(x); 
5. explicit keyword: inhibit implicit calls to constructors.
6. Operator extension (special cases):
   Subscript: T& operator [] (S index)
   Arrow: X* operator -> ()
   Function call: T2 operator () (arg list)
   Cast: operator T()

Exceptions
1. An event that prevents normal continuation.
2. May due to program errors, data errors, or external events, such as file not found, insufficient permissions, network failure, read error, and out of memory error.
3. Exception handling: try again, try something else, or give up.
4. C-style status values or error codes.
6. Throw an exception. E.g., throw AgeError("Age can’t be negative");
7. Try a block and catch possible exceptions. E.g.,
   try { ... // run some code
   catch (AgeError& aerr) {
   // report error
   cout<< "Age error: ";
   aerr.printError( cout )<< cout;
   ... // recover or abort
   }

8. Standard exception class std::exception. Polymorphic and contains a virtual function const char* what() const;
9. Multiple catch blocks.
10. Rethrow: {throw e; } vs. {throw; }
11. Termination: exception not caught by any catch block, or a destructor issues an throw during stack unwinding (in which case terminate() is called).
12. One-sentence conclusion: all exceptions should be caught and dealt with explicitly.

Design Patterns
2. What is a design pattern: a pattern name, the problem, the solution, the consequences.
3. Some popular design patterns:
   (a) Adaptor: match interface between toolkit class and application.
   (b) Indirection: decouple application from implementation depending on low-level devices.
   (c) Proxy: similar to indirection, control access via a placeholder.
   (d) Polymorphism: multiple implementation from the abstraction.
Final Exam Review Topics

(e) Controller: handling system events.
(f) Bridge: generalization of indirection, used when both application and implementation are polymorphic.
(g) Subject-observer (publish-subscribe): keep updated states of the subject with observers.
(h) Singleton: enforce one instance of a class.
(i) Strategy: define a family of algorithms, encapsulate each one, and make them interchangeable (at runtime).
(j) Factory method: define an interface for creating an object but let the implementation decides which class to instantiate.

Graphical User Interfaces

1. Command line vs. GUI.
2. Standard interfaces for C++: iostream and associated packages.
3. GTK+ (gtkmm) and Qt.
4. Structure of a GUI: windows, widgets, and a widget tree.
5. Concurrency and events (event loop, event handler).
6. Binding system calls to user functions, polymorphic binding, callback registration, signals and slots.
7. Create a GUI using gtkmm: initialize gtkmm, create a window, create and lay out widgets within the window, connect user code to events, show the widgets, enter the main event loop.

Demos

- Please review demos.

Good luck for the final!