

Final Review

CS2704: Object-Oriented Software Design
and Construction

Constantinos Phanouriou
Department of Computer Science
Virginia Tech

CS2704 (Spring 2000)

1

Final Exam

Tuesday, May 9 (1:05pm – 3:05pm)

Davidson 3

Format:

- Some fill-in, multiple choice, true/false
- Some short answer
- Some coding questions

Exam will be “Close book and close notes”

CS2704 (Spring 2000)

2

Topics

- Design
 - Class diagram notation
 - Represent class relationships
 - Aggregation, Association, Inheritance
 - Polymorphism
- C++
 - Describe the anatomy of a C++ class
 - Access control (private, protected, public)
 - Scope and lifetime: automatic vs. dynamic memory
 - Use of modifiers: const, static
 - Use constructors and destructors to manage problems with dynamic memory (deep copy vs. shallow copy)
 - Standard Template Library (STL)
 - Templates
 - Inheritance
 - Polymorphism: virtual functions, pure virtual functions, abstract classes
 - Exceptions

CS2704 (Spring 2000)

3

Object-Oriented Design Strategies

- **Abstraction** – modeling essential properties
- **Separation** – treat what and how independently
- **Composition** – building complex structures from simpler ones
- **Generalization** – identifying common elements

CS2704 (Spring 2000)

4

Composition (1)

- **Composition:** An organized collection of components interacting to achieve a coherent, common behavior
- There are two forms of composition:
 - Association (or acquaintance)
 - Aggregation (or containment)

CS2704 (Spring 2000)

5

Composition (2)

- The two forms of composition are similar in that they are both part-whole constructors
- What distinguishes them is the visibility of the parts
 - In an aggregation, only the whole is visible and accessible
 - In association, the interacting parts are externally visible and may be shared by different compositions

CS2704 (Spring 2000)

6

Aggregation

- Aggregation describes a structure in which one component, the whole, contains the other components, the parts
- The enclosing object uses the functionality provided by the parts to implement its own behavior
- Types: static and dynamic aggregation

CS2704 (Spring 2000)

7

Association

- Association is a part-whole organization in which the whole is exactly defined by the parts and the relationships among the parts
- Each part of the composition maintains its identity, external visibility, and autonomy in the composition

CS2704 (Spring 2000)

8

Generalization

- Find and exploit the common or shared aspects in a set of abstractions
 - Hierarchy
 - Polymorphism
 - Genericity
 - Patterns

CS2704 (Spring 2000)

9

Inheritance

- Inheritance expresses sharing among classes
- Classes may share
 - a common implementation (code and/or data)
 - a common interface (implement same methods)
 - or both

CS2704 (Spring 2000)

10

Composition Strategies (P. Coad)

- Composition = aggregation and association
- *Composition Strategy*: Use composition to extend responsibilities by delegating work to other classes.
- Prefer composition over inheritance

CS2704 (Spring 2000)

11

Inheritance Strategy (Coad)

- Inheritance is used to extend attributes and methods
- Use should be restricted, because the relationship between base and derived classes leads to a weak form of encapsulation

CS2704 (Spring 2000)

12

When to Use Inheritance

- Inheritance relationship must satisfy:
 1. Represents “is a special kind of”, and not “is a role of”
 2. An object of one class in hierarchy never needs to transmute to another class
 3. Derived class extends rather than overriding or nullifying base class
 4. Does not derive for the purpose of copying useful capabilities
 5. If classes from problem domain, represents special kinds of roles, transactions or devices

CS2704 (Spring 2000)

13

Polymorphism

- “The ability to manipulate objects of distinct classes using only knowledge of their common properties without regard for their exact class”
- Generality is achieved by allowing the algorithm to uniformly manipulate objects of different
 - (provided the algorithm uses only properties shared by the different classes)

CS2704 (Spring 2000)

14

Class Interface Declaration

```
class Frame {  
    public:  
        // interface visible to the user goes here  
    protected:  
        // interface visible to subclasses goes here  
    private:  
        // hidden declarations go here  
};
```

CS2704 (Spring 2000)

15

Access Control

- public:
 - Declare interface (usually only methods)
 - Usable anywhere outside of class
- protected:
 - Declare common attributes or methods common in all subclasses
- private:
 - Prevent access outside of class
 - Primarily attributes (data), some methods

CS2704 (Spring 2000)

16

Taxonomy of member functions

- Member functions implement operations on objects
- Here is one common taxonomy:
 - **Constructor** – an operation that creates a new instance of a class (i.e., an object)
 - **Mutator** – an operation that changes the state of one, or more, of the data members of an object
 - **Observer** (reporter) – an operation that reports the state of one or more of the data members of an object, without changing them
 - **Iterator** – an operation that allows processing of all the components of a data structure sequentially

CS2704 (Spring 2000)

17

C++ Issues

- Inheritance
- Polymorphism (virtual functions)
- Templates
- STL
- Exceptions

CS2704 (Spring 2000)

18

Inheritance

```
class subClass : public baseClass {  
    private:  
  
    public:  
  
};
```

CS2704 (Spring 2000)

19

Inheritance

- Inheritance: public, protected, private
- Constructor execution order
- Dealing with inherited methods
 - replacing, extending, hiding
- Casting

CS2704 (Spring 2000)

20

Virtual methods

- Methods can be declared as virtual
- Sets up dynamic binding mechanism
 - Use pointer for base class to point to object of derived class (like before)
 - Method call for virtual method is dynamically bound to method of derived class
- Methods declared as virtual in base class are virtual for all derived classes

CS2704 (Spring 2000)

21

Pure Virtual Methods

- A *pure virtual method* has a null definition
virtual void draw(Canvas&) = 0;
- *Abstract class* - class in which **at least one** method is pure virtual
 - Cannot be instantiated (no objects)
 - Can have fields and other methods
- *Pure abstract class* - defines an interface

CS2704 (Spring 2000)

22

Template Syntax

- Can specify one or more type parameters

```
template <class Item> class list { ... };  
  
template <class Item, class ItemCompare>  
void sort(const list<Item>& l) { ... }
```

Parameterized class

Template parameter

CS2704 (Spring 2000)

23

Template Example

```
template <class BlahBlah> class Queue {  
private:  
    BlahBlah buffer[100];  
    int head, tail, count;  
public:  
    Queue();  
    void Insert(BlahBlah item);  
    BlahBlah Remove();  
    ~Queue();  
};
```

Keywords

Parameter, can be any type

Either a class or method definition starts here.

CS2704 (Spring 2000)

24

Templates

- Definition of “template”:
Parameterized class with parameters.
Parameters denote unknown type.
- Usage:
Situations where *same* algorithms & data structures are applied to *different* data types
- Syntax Example:

```
template <class BlahBlah> class Queue {...}
```

CS2704 (Spring 2000)

25

Instantiating a template

- Given:

```
template <class BlahBlah> class Queue {...}
```
- Instantiate Queue of ints in 2 ways:
 - `Queue<int> intQueue;`
 - `typedef Queue<int> IntegerQueue;`
`IntegerQueue intQueue;`

Both of these define an **object** intQueue.

CS2704 (Spring 2000)

26

Constant Parameter

- Special cases are sometimes desired

```
class Queue<float> {  
private:  
    float buffer[100];  
    int head, tail, count;  
public:  
    Queue();  
    void Insert(float item);  
    float Remove();  
    int Size();  
    ~Queue();  
};
```

CS2704 (Spring 2000)

27

What is STL?

- STL (or Standard Template Library) is a general-purpose library of:
 - generic algorithms and
 - data structures
- Makes programmer more productive:
 - Contains a lot of different components that can be plugged together and used in an application
 - Provides a framework into which different programming problems can be decomposed

CS2704 (Spring 2000)

28

STL Components

- **Containers** - classes for objects that contain
- **Generic algorithms** - functions that work on different types of containers
- **Iterators** - “pointers” into containers
- **Function objects**
- **Adaptors** - classes that “adapt” other classes
- **Allocators** - objects for allocating space

CS2704 (Spring 2000)

29

Sequence Containers

- **Arrays** - random access, fixed length, constant time access
- **vector<T>** - random access, varying length, constant time insert/delete at end
- **deque<T>** - random access, varying length, constant time insert/delete at either end
- **list<T>** - linear time access, varying length, constant time insert/delete anywhere in list

CS2704 (Spring 2000)

30

Maps and Multimaps

- Associative arrays on given Key type
- Map requires unique Keys (by def of order)
- Multimaps allows duplicate Keys
- Map is like set that holds key-value pairs, which are only ordered on the keys
- Additional operator: `map1[k] = v`

CS2704 (Spring 2000)

31

Choosing Container

- Vector used in place of dynamically allocated array
- List allows dynamically changing size for linear access
- Set used when need data kept sorted
- Map used when want indexed data
- Multi(set/map) when need multiple keys

CS2704 (Spring 2000)

32

C++ Exceptions

- Exception based on idea that if error happens don't want to continue with normal flow
- Two parts to exceptions
 - throw (or raise)
 - catch (or handle)
- Separates error reporting from error handling

CS2704 (Spring 2000)

33

Exception Handler

- Try block: `try { /* statements */ }`
- Handler: `catch (/* class name */) { /* ...*/ }`
- Handlers
 - must follow try block, or another handler
 - Can throw “caught” exception - do something and then decide must be handled further up call chain
- Catch-all: `catch (...) { /* code */ }`

CS2704 (Spring 2000)

34