

Midterm Review

CS2704: Object-Oriented Software Design and Construction

Constantinos Phanouriou
Department of Computer Science
Virginia Tech

CS2704 (Spring 2000)

1

Outline

- Theory
 - Object-oriented design
 - Composition
 - Class Design and Evaluation
- C++
 - C++ Classes
 - Access control
 - Overloading

CS2704 (Spring 2000)

2

Object-Oriented Design

- What are the differences between:
 - procedural programming
 - divide problem into a sequence of subproblems to be solved
 - program is a sequence of procedure calls
 - object-oriented programming
 - divide problem into parts that interact to produce the solution
 - program is a collection of objects that interact

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

Abstraction

- Design technique that focuses on the essential aspects of an entity and ignores or conceals less important or nonessential aspects
- Properties of a good abstraction
 - well named – clearly identifies abstraction
 - coherent – sensible description
 - accurate – only attributes of entity
 - minimal – no irrelevant attributes
 - complete – everything needed

CS2704 (Spring 2000)

5

Separation

- Separate the *what* is to be done from *how* it is to be done
- Define classes by independently specifying the interface for objects in that class, and the implementations of that interface

CS2704 (Spring 2000)

6

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)

7

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)

8

Example

- Aggregation – Soda machine
 - It is a whole composed of several internal parts (cooling system, coin acceptor, change maker, soda supply). These parts are not visible or accessible to the normal user.
- Association – Computer Station
 - The workstation consists of a keyboard, a mouse, a monitor, and a processor. Each of these interacting parts are visible to the user and can be directly manipulated by the user

CS2704 (Spring 2000)

9

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

CS2704 (Spring 2000)

10

Advantages of Aggregation (1)

- **Simplicity:** The aggregating class or object allows the entire assembly encapsulated subobjects to be referred to as a single unit
- **Safety:** Through encapsulation the subobjects are protected from accidental misuse by outside elements
- **Specialization:** The public interface provides operations that apply to several of the subobjects as a group. Can also provide more meaningful names

CS2704 (Spring 2000)

11

Advantages of Aggregation (2)

- **Structure:** The existence of the encapsulating boundary captures the designer's intent, that all objects function as a unit
- **Substitution:** An alternative implementation of the object defined by aggregation can be substituted without affecting other parts of the system as long as the public interface of the aggregating object remains unchanged

CS2704 (Spring 2000)

12

Types of Aggregation

- Two types of aggregation:
 - Static aggregation
 - The lifetimes of the subobjects are identical to the lifetimes of the containing object
 - The subobjects are explicitly declared in the class of the containing object
 - Dynamic aggregation
 - At least some of the objects known only to the containing object are created dynamically, via the new operator, at run-time
- Example: An automobile has a fixed number of tires, but a tire store has a variable number of tires

CS2704 (Spring 2000)

13

Example: static aggregation

```
class Automobile {  
    private:  
        Tire tires[4];  
  
    public:  
        Automobile();  
        void Drive(Location loc);  
}
```

CS2704 (Spring 2000)

14

Example: dynamic aggregation

```
class TireFactory {  
    private:  
        Tire *tires;  
  
    public:  
        TireFactory(int numTires) {  
            tires = new Tire[numTires];  
        }  
        void addTire(int num);  
        void deleteTire(int num);  
}
```

CS2704 (Spring 2000)

15

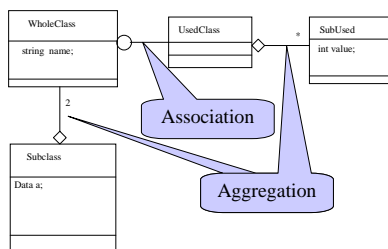
Association

- 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)

16

Class Design



CS2704 (Spring 2000)

17

Evaluating the Design

- **Behavioral:**
 - Emphasizes *actions* in system
- **Structural:**
 - Emphasizes relationships among components
- **Information:**
 - Emphasizes role of information/data/state and how it's manipulated

CS2704 (Spring 2000)

18

Class Interface Declaration

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

CS2704 (Spring 2000)

19

Access Control

- public:
 - Declare interface (usually only methods)
 - Usable anywhere outside of class
- private:
 - Prevent access outside of class
 - Primarily attributes (data), some methods

CS2704 (Spring 2000)

20

Constructors

- Responsible for initializing new objects
- Default: `CS_Class()`;
- Copy: `CS_Class(const CS_Class& c)`;
- User:
`CS_Class(int nindex);`
`CS_Class(int nindex, char* name);`

CS2704 (Spring 2000)

21

Constructors and Initialization

- Sequence of object creation:
 1. Create storage for object
 2. Initialize storage
 3. Execute body of constructor

CS2704 (Spring 2000)

22

Default Constructor

- If you do not provide a constructor method, the compiler will automatically create one
- The default constructor:
 - takes no arguments
 - is called for each data member that is an object of another class
 - provides no initialization for data members that are not objects
- Always implement your own default constructor

CS2704 (Spring 2000)

23

Destructors

- Responsible for properly destroying object
- Prototype: `~CS_Class()`;
- Declare one even if you don't need it:
`~CS_Class() {}`
- Important when have pointers as field
- Destructors cannot be static

CS2704 (Spring 2000)

24

Matters of Style

- One class to one pair of files
- Use class name as file name
- Public first, private second (for class user)
- Only prototypes in class declaration
- Function definitions in implementation file

CS2704 (Spring 2000)

25

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)

26

Function Overloading

- In C++ it is legal, although not always wise, to declare two or more functions with the same name.
- Compiler determines which function to call as follow:
 1. Exact match (no conversions or only trivial ones like array name to pointer)
 2. Match using promotions (bool to int; char to int; float to double, etc.)
 3. Match using standard conversions (int to double; double to int; etc.)
 4. Match using user-defined conversions
 5. Match using the ellipsis . . . in a function declaration (ditto)

CS2704 (Spring 2000)

27

Inline Functions

- Generally more efficient for *small* functions
- Expanded in-place of invocation
 - Eliminates method invocation overhead
 - Compiler generates necessary code and maps parameters automatically
 - Still only one copy of function implementation
- Two ways to specify an inline method
 - Provide implementation during class definition (default inline)
 - Use 'inline' keyword (explicit inline)

CS2704 (Spring 2000)

28

this

- A predefined variable, provided automatically, which is a pointer to the object itself

CS2704 (Spring 2000)

29

Operator Overloading (1)

- **Operator overloading** is the ability to define a new meaning for an existing operator
- Each operator has predefined and unchangeable meaning for the built-in types (int, float, char, etc.)
- Each operator can be given a specific interpretation for individual user-defined classes or combination of user-defined classes
- Compiler recognizes which function to use by signature (types of arguments)

CS2704 (Spring 2000)

30

Operator Overloading (2)

- There are a number of reasons why a class designer may decide to provide extensions to one or more of the built-in operators:
 - Support natural, suggestive usage
 - Semantic integrity
 - Uniformity with built-in types
- Function that modify the meaning of operators can be defined:
 - as function members of a class
 - as non-member function

CS2704 (Spring 2000)

31

Using Overloaded Operators

- If `operator==` defined as *member* function
`nme1 == nme2`
is the same as
`nme1.operator==(nme2)`
- If `operator==` defined as *nonmember* function
`nme1 == nme2`
is the same as
`operator==(nme1, nme2)`

CS2704 (Spring 2000)

32

Using non-member functions

- There are two situation under which operator overloading must be done by functions that are not members of a specific class
- When the class to which the member function should be added is not available for modification
 - Typical for standard library classes (e.g., I/O streams)
- When type conversion of the arguments involved in the operation is desired

CS2704 (Spring 2000)

33

Prefix and Postfix Operators

- A prefix operator
`Day operator++(); //member`
`Day operator++(Day&); //nonmember`
- A postfix operator
`Day operator++(int); // int is dummy`
`Day operator++(Day&, int);`
- The `int` is a dummy type to show postfix

CS2704 (Spring 2000)

34

Modifiers (1)

- An ordinary member function declaration specified three logically distinct things:
 1. The function can access the private part of the class declaration, and
 2. the function is in the scope of the class, and
 3. the function must be invoked on an object (has a `this` pointer)`void add(int x);`

CS2704 (Spring 2000)

35

Modifiers (2)

- A static member function declaration specified two logically distinct things:
 1. The function can access the private part of the class declaration, and
 2. the function is in the scope of the class, and
 3. ~~the function must be invoked on an object (has a `this` pointer)~~`static void add(int x);`

CS2704 (Spring 2000)

36

Modifiers (3)

- A friend member function declaration specified one logically distinct things:
 1. The function can access the private part of the class declaration, and
 - ~~2. the function is in the scope of the class, and~~
 - ~~3. the function must be invoked on an object (has a this pointer)~~

```
friend void add(int x);
```

CS2704 (Spring 2000)

37

Type of conversions

- Implicit conversion
 - compiler is responsible for determining that a conversion is needed and how to perform it
- Explicit conversion
 - programmer assume full responsibility
 - two different, but equivalent, syntaxes for explicit conversion

CS2704 (Spring 2000)

38

Conversion Uses

- Type conversion is needed for :
 - resolving mismatched types in assignments and expressions
 - when passing parameters to functions
- The existence of type conversions makes it possible to use one type when a different type may be expected

CS2704 (Spring 2000)

39

Type Conversion Operators (1)

- Constructors can play a role in type conversion
- A constructor can be viewed as a way of converting one data type to another
- Example:

```
class Date {
public:
    Date(string);
    // ...
}
```

CS2704 (Spring 2000)

40

Type Conversion Operators (2)

- Constructors cannot specify:
 - an implicit conversion from a user-defined type to a basic type (because the basic type are not classes), or
 - a conversion from a new class to a previously defined class (without modifying the declaration for the old class)
- These problems can be solved by defining a conversion operator for the source type

CS2704 (Spring 2000)

41

Conversion Operator

- A conversion operator is a member function:
 - `X::operator T()`, where T is a type name, defines a conversion from X to T. No return value.
- Example:

```
class Date {
public:
    Date(string);
    operator string();
    // ...
}

Date::operator string(){
    string s;
    //...
    return s;
}
```

CS2704 (Spring 2000)

42

Controlling Change

- Protecting the integrity of an object and limiting the ways in which it may be modified
- C++ Keyword: **const**
 - Any attempt to modify the value of a **const** object after its initializing declaration will result in a compiler-time error
 - When an object is declared as **const**, then only methods declared as **const** can be called

CS2704 (Spring 2000)

43

Pointers and Constant

```
char s[] = "Gorm"; char *p = 0;

const char* pc = s; // pointer to const
pc[3] = 'g';       // error: pc points to constant
pc = p;           // ok

char *const cp = s; // constant pointer
cp[3] = 'a';       // ok
cp = p;           // error: cp is constant

const char *const cpc = s; // const ptr to const
cpc[3] = 'a';           // error: cpc points to constant
cpc = p;               // error: cpc is constant
```

CS2704 (Spring 2000)

44