C++ Classes

CS2704: Object-Oriented Software Design and Construction

Constantinos Phanouriou
Department of Computer Science
Virginia Tech

Outline

- Class Interface Declaration
- Access Control
- Constructors and Destructors
- Instance Declaration

Class Interface Declaration

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

Access Control

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

Class Example

class CS_Class {
public:
    int getIndex() { return index; }
    void setIndex(int i) { index = i; }
private:
    int index;
};

Class Example (cont.)

void main() {
    CS_Class cs2704;
    cs2704.setIndex(5454);
    cout << cs2704.getIndex();
}
void main() {
    CS_Class cs2704;
    cs2704.index=5454; // Error: cannot access private member
    cout << cs2704.getIndex();
}

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);

Instance Declaration

- Use class name like built in type
- Declaration uses different constructors
  - CS_Class cs2704; // uses default
  - CS_Class cs2704_00(cs2704); // copy
  - CS_Class cs2704_5454(5454); // user
  - CS_Class cs2704_1373(1737, "CS2704 8T"); // user
- When decl. comes in scope, object created

Class Example (with constructor)

class CS_Class {
public:
    CS_Class() {}
    CS_Class(int nindex) { index = nindex; }
    int getIndex() { return index; }
    void setIndex(int i) { index = i; }
private:
    int index;
};

void main() {
    CS_Class cs2704(5454);
    cout << cs2704.getIndex();
}

Class Example (copy constructor)

class CS_Class {
public:
    ... CS_Class(const CS_Class & c) { ... }
    ...
private:
    int index;
};
Class Example (copy constructor)

```cpp
void main() {
    CS_Class cs2704(5454);
    CS_Class mySection(cs2704);
    cout << mySection.getIndex();
}
```

Constructors and Initialization

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

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

Class Example (with destructor)

```cpp
class CS_Class {
    public:
        CS_Class() {}
        CS_Class(int nindex) {index = nindex;}
        ~CS_Class() {};
        int getIndex() { return index; }
        void setIndex(int i) { index = i; }
    private:
        int index;
};
```

Destructor

- 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

Destructor Example

```cpp
class CS_Class {
    public:
        CS_Class() {name = new char[100];}
        ~CS_Class() {delete[] name;}
        ...
    private:
        char * name;
        ...
};
```
Class Methods

- Declare prototypes in class declaration
  ```
  bool isSection(int index);
  ```
- Note: no return type for constructors and destructors
- Using methods
  - Inside class: `isSection(5454);`
  - Outside class: `cs2704.isSection(5454);`

Parameters

- Pass-by-value
  ```
  bool hasName(string name);
  ```
- Pass-by-reference
  ```
  bool hasName(string& name);
  ```
- What happens when pass complex object by value?

Decl. And Def. Example

```
class CS_Class {
public:
    int getIndex();
    void setIndex(int i);
private:
    int index;
};
```  

Decl. And Def. Example (cont.)

```
int CS_Class::getIndex() {
    return index;
}
void CS_Class::setIndex(int i) {
    index = i;
}
```  

const and Methods

- Proper use of `const` allows compiler to help find errors
- Method that doesn’t change object
  ```
  bool isSection(int index) const;
  ```
- Method that doesn’t change ref parameter
  ```
  void setName(const string& nm);
  ```
- Method that gives reference to internal object
  ```
  const string& name() const;
  ```

const Method Example

```
class CS_Class {
public:
    int getIndex() const;
    void setIndex(int i);
private:
    int index;
};
```
Assignment Method

- Overloaded operator
  `CS_Class& operator=(const CS_Class&);`
- Usually similar to copy constructor
- If have pointers as fields
  - Check not doing something like `a = a;`
  - Delete pointer values
  - Copy values
- End with `return *this;`

Matters of Necessity

If you want to use your class like a built-in type, you must include
- A default constructor
- A copy constructor
- An assignment operator
- A destructor

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
  More advanced language features force us to break some of these rules