Introduction to classes

William Hendrix

Lecture 15

Outline

- Object-oriented programming
- Defining classes
- Constructors
- Destructors
- Member functions

Object-oriented programming

- Up until now, we've mainly thought about what our programs *do* and how they *act*
  - Focus on problem verbs to identify functions
- Object-oriented programming focuses on *objects* (nouns) and how they interact
  - Accomplished in C++ through the use of classes
- Classes
  - Encompasses data and functions that allow the class objects to interact with the rest of the program
  - Example: `string`, `str.length()`
  - By *encapsulating* data, you only allow it to be modified in valid ways
    - Reduces chance for bugs in code
  - Well-designed classes also promote code reuse

Defining classes

- Syntax:

```
class Class_name
{
  private:
  int data_member1;
  //...
  public:
  void member_function1();
};
```

- **Convention:** class names are capitalized
- Data members and member functions can be any type
  - Special function types: constructors and destructors
- **Pitfall:** don't forget the semicolon
- Access modifiers
  - Public: anyone can call or modify
  - Private: only the class can access
    - You shouldn't declare public data members without good reason

Constructors

- Function called when object is created

```
string str;
string str("initial value");
```

- Goal: ensure that all data members are initialized to some meaningful value
- Syntax:

```
public:
Class_name();
```

- Can have any number of arguments
  - Classes may have multiple constructors with different arguments
- Classes should almost always have a default (no argument) constructor
- Additional constructors can initialize data members with provided values
- Note: constructors are *only* called when creating objects

Destructors

- Function called when object is destroyed
  - E.g., when local variable passes out of scope
  - Free allocated memory, close files, etc.
- Syntax

```
public:
//...
virtual ~Class_name();
```

- We will discuss `virtual` much later…
- Must have no parameters
  - Only one destructor allowed
- Default destructor
  - If no destructor specified, a trivial destructor is used
  - Also true for constructors, but using default constructor is **very** bad style

Accessors and mutators

- Common function types
- Mutators change class data members
  - Should always check input validity
  - E.g., `void setYear(int year);`
- Accessors report or compute values without changing data members
  - Generally followed by `const` keyword
    - Denotes a function that can be called on a `const` object
  - E.g., `int getYear() const;`
- Common functions: `getX` and `setX`

Implementing functions

- Syntax

```
void Class_name::member_function(int arg)
{
  //Code goes here
}
```

- Scope resolution operator (`::`)
- Name, return type, and arguments must match class definition
- Can access or modify any `private` data members or functions as though they were local
- Example

```
void Date::setYear(int newYear)
{
  if (newYear == 0)
    year = 1;
  else
    year = newYear;
}
```

Implementing constructors/destructors

- Basic syntax

```
Class_name::Class_name()
{
  //Initialize data members
}

virtual Class_name::~Class_name()
{
  //Clean up data
}
```

- No return type

Class design

- What does class represent?
  - What data members does it need?
- How are users going to interact with the class?
  - How can they construct the class?
    - What should the default constructor do?
  - What should they be able to do or calculate with class?
    - What are the common or important operations?
    - How does this class interact with other classes or primitive data types?
    - What data members have an accessor?
  - How should they be allowed to modify an object?
    - What errors can they make?
    - How do you handle errors?
  - What clean up needs to be done when the object is destroyed?

Exercise

- Write a class that encapsulates a complex number. Your Complex numbers should be able to:
  - Print themselves
  - Multiply or add another Complex number
    - Return the result, don't mutate
  - Calculate their norm $(|a + bi| = \sqrt{a^2 + b^2})$

- Complex should have a default constructor as well as a constructor that accepts real and imaginary parts as `double`

Tonight

Recommended reading: Sections 5.1-5.7