CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

October 30, 2017

Outline

Outline

Class Syntax

Basics

- C++ is Object-Based: classes can be implemented.
- Classes have the same components as in Python
- Data Members = Attributes = (Instance or Class) Variables
- Member Functions = (Instance or Class) Methods
- C++ is compiled, so classes (with their components) must be declared before they can be used by any code.
- A class declaration is usually written in a header file (`<classname>.h`), so it can be included by any program file using or implementing the class. It declares data members and member functions.
- The definitions of member functions of a class are usually in an implementation file (`<classname>.cpp`). Once it is compiled, it can be linked with any application file.

Class Syntax

Form of a Class Declaration (Definition)

- Begins with `class <classname> {`
- Data Member declarations are similar to those for local variables: type, name.
- Member Functions declarations are similar to those for normal functions: return type; function name; parameter list; default values.
- Does not (usually!) give the implementation of member functions.
- Ends with };

Class Syntax

Member Functions

- No `self` parameter!
- Like function declarations: return type; function name; parameter list; default values.
- `const` member functions do not change any data member (attribute) of the object.
- `inline` member function declarations have implementation code.
- Constructors have the same name as the class. Different constructors for one class must have different signatures (formal parameter lists).
- (later: destructors)

Class Syntax

Access Specifiers

Set access levels for member functions and data members:

- `public:` available outside the scope of the class.
- `private:` available only within the scope of the class.
- `protected:` available within the scope of the class or within subclasses derived from the class.

Class Syntax

Class Header File Example: rational.h

```
#ifndef _RATIONAL_H
#define _RATIONAL_H
class Rational {
public:
   Rational(int n = 0, int d = 1) { set(n, d); }
   bool set(int n, int d);
   int num() const { return num_; }
   int den() const { return den_; }
   double decimal() const {return num_/double(den_);}
private:
   int num_, den_;
};
#endif
```

Class Syntax

Class Implementation Files (.cpp)

- Include the header file for that class
- Class Implementation
- Member Function Implementations
- Scope resolution operator (::): (classes and namespaces)

Class Syntax

Class Implementation File Example: Rational.cpp

```
#include "Rational.h"
bool Rational::set(int n, int d)
{
   if (d != 0) {
      num_ = n;
      den_ = d;
      return true;
   }
   else
      return false;
}
```

The C++ string Class

Usage

- `#include <string>` to use this class.
- `<<`, `>>` are overloaded to work with `cin` and `cout`.
- `getline(cin, name)`: all text typed before the end-of-line delimiter goes into `name`.
- `<, <=, >, >=, ==, !=, +, +=` are overloaded.

IFSTREAM, OFSTREAM CLASSES

Usage

- `#include <fstream>` to use these classes.
- Use `infile.open(filename.c_str())`, `outfile.open(filename.c_str())` to access.
- <<, >> are overloaded to work with `ifstream` and `ofstream`.
- `getline(cin, name)` gives name all text typed before the end-of-line delimiter.
- Use `infile.close()`, `outfile.close()` to end access.

Overloading Operator Symbols

Overloading '+' as Standalone Function

```
class Rational {
public:
   Rational(int n = 0, int d = 1) { set(n, d); }
   ...
   // access functions
   int num() const { return num_; }
   int den() const { return den_; }
   ...
private:
   int num_, den_;
};
Rational operator+(const Rational &r1, const Rational &r2);
```

Overloading Operator Symbols

Overloading '+' as Standalone Function

```
...
Rational operator+(const Rational &r1, const Rational &r2)
{
   int num, den;
   num = r1.num() * r2.den() + r1.den() * r2.num();
   den = r1.den() * r2.den();
   return Rational(num, den);
}
```

Overloading Operator Symbols

Overloading '+' as Standalone Function

```
// mainv1.cpp
# include "Rationalv1.h"
int main()
{
   Rational r1(2, 3), r2(3, 4), r3;
   r3 = r1 + r2; // common method of calling
   r3 = operator+(r1, r2); // direct method of calling
}
```

Overloading Operator Symbols

Overloading '+' as Member Function

```
class Rational {
public:
   Rational(int n = 0, int d = 1) { set(n, d); }
   ...
   // access functions
   int num() const { return num_; }
   int den() const { return den_; }
   ...
   Rational operator+(const Rational &r2) const;
private:
   int num_, den_;
};
```

Overloading Operator Symbols

Overloading '+' as Member Function

```
...
Rational Rational::operator+(const Rational &r2) const
{
   Rational r;
   r.num_ = num_ * r2.den_ + den_ * r2.num_;
   r.den_ = den_ * r2.den_;
   return r;
}
...
```

Overloading Operator Symbols

Overloading '+' as Member Function

```
   // mainv2.cpp
   #include "Rationalv2.h"
int main()
{
   Rational r1(2, 3), r2(3, 4), r3;
   r3 = r1 + r2; // common method of calling
   r3 = r1.operator+(r2); // direct method of calling
}
```

Overloading Operator Symbols

friend Functions and Classes

- Declared within the definition of a class using the keyword `friend`
- Allowed to have access to the private data and functions of the class.
- Needed for efficient performance with other classes.

Overloading Operator Symbols

friend Example: Rational.h

```
friend std::istream& operator>>(std::istream& is, Rational &r);
friend std::ostream& operator<<(std::ostream& os, const Rational
&r);
...
std::istream& operator>>(std::istream &is, Rational &r);
std::ostream& operator<<(std::ostream &os, const Rational &r);
```

Overloading Operator Symbols

friend Example: Rational.cpp

```
std::istream& operator>>(std::istream &is, Rational &r)
{
   char c;
   is >> r.num_ >> c >> r.den_;
   return is;
}
std::ostream& operator<<(std::ostream &os, const Rational &r)
{
   os << r.num() << "/" << r.den();
   return os;
}
```

Class Variables

Syntax

- Declared using the `static` keyword.
- All instances (objects) in the class share the same value for a class variable. There is only one value for the entire class.
- Just as in the Python `Card` class.

Class Variables

Example: Card.h

```
class Card {
   ...
private:
   ...
   static const std::string suits_[4];
   static const std::string faces_[13];
};
```

Class Methods

Syntax

- Declared using the `static` keyword.
- Can only access class variables. (A function call to a class method is not related to any particular instance.)
- Can be used to count how many instances are alive for a class: increment count in the constructor, decrement the count in the destructor.
- Must call using the class name and scope qualifier: `Card::count()`.