static Class Members and Operator Overloading

Object Details

- What does memory look like after creating multiple objects of a class?
- For example:
  - Time t(3, 45, 00);
  - Time t2(5, 29);
  - Time t3(14);
  - Time t4;
  - Time *pTime = new Time();

static Class Members

- Each object gets it's own copy of the data members
- What if we wanted a data member to be shared between all objects
  - Each object sees the same value for the data member
  - Each object can modify that data member, and the other objects will see the change
- Data members of this type are called static

static Class Member

- static members represent class-wide information and are not specific to one object
- There is only one copy of the member and it is shared between all objects
- Why would we ever need or want a static class member? Can you think of an example.

static Class Members

- They are not global variables
- The static data member could be declared public, private, or protected
- static data members must be initialized once

Example

```
#ifndef EMPLOYEE2_H
#define EMPLOYEE2_H
class Employee
{
public:
   Employee( const char *, const char * );
   ~Employee();
   const char *getFirstName() const;
   const char *getLastName() const;
   static int getCount();
private:
   char *firstName;
   char *lastName;
   static int count;
};
#endif
```

Information Hiding and ADTs

- ADT: Abstract Data Type
- The IntegerSet class we looked at last week is a prime example of an ADT
  - Hide the implementation from the client
    - I.e. Clients don't need to know that a set is implemented as an array, where the indexes of the array represent the elements in the set
    - All the clients want to do is use the ADT for their programs
- How could we change the implementation of the IntegerSet ADT but still provide the same functionality to the client?

ADT

- An ADT captures two notions:
  - Data representation
  - Operations allowed on the data
- Examples of ADTs
  - Array ADT
  - String ADT

Operator Overloading

- A couple of weeks ago we created a class for rational numbers
- An example of how a client would use that class is:

```
Rational a( 3, 4 );
Rational b( 2, 5 );
Rational c, d;
c = a.multiplication( b );
d = a.addition( b );
```

- It would be much easier if we could instead write

```
c = a * b;
d = a + b;
```


Operator Overloading

- We defined a print function for the IntegerSet class to output the contents of a set

```
IntegerSet setA( 10 );
setA.print();
```

- Wouldn't it be more efficient and more consistent with C++ if we could write

```
cout << setA;
```


The How of Operator Overloading

- Write a function definition for the operator, but the function name becomes **`operator`** followed by the symbol
  - operator<<
  - operator+
  - operator==
- Two operators are used without overloading
  - & the address operator
  - = memberwise assignment

Operator Overloading

- Operator overloading can be achieved in one of two ways
  - A member function of the class
  - A friend function of the class
- Using operator overloading through member functions has the restriction that the object of the class must always be to the left of the operator
  - Not useful for the insertion operator <<

operator<<

- << must be overloaded using friend functions
- The return value of operator<< is an ostream&
- The arguments will be the output stream and an object of the class

Example

```
class PhoneNumber
{
  friend ostream &operator<<( ostream&, const
  PhoneNumber & );
  friend istream &operator>>( istream&, PhoneNumber &
  );
private:
   char areaCode[ 4 ];  // 3-digit area code and null
   char exchange[ 4 ];  // 3-digit exchange and null
   char line[ 5 ];      // 4-digit line and null
}; // end class PhoneNumber
```


Definition

```
ostream &operator<<( ostream &output, const
  PhoneNumber &num )
{
   output << "(" << num.areaCode << ") "
          << num.exchange << "-" << num.line;
   return output;     // enables cout << a <<
  b << c;
} // end function operator<<
```


Driver

```
int main()
{
   PhoneNumber phone;
   cout << "The phone number is: ";
   cout << phone << endl;
   return 0;
} // end main
```


Your Turn

```
class Rational
{
public:
   Rational( int = 0, int = 1 );
   Rational addition( const Rational & );
   Rational subtraction( const Rational & );
   Rational multiplication( const Rational & );
   Rational division( const Rational & );
   void printRational ();
private:
   int numerator;
   int denominator;
   void reduction();
};
```

- Replace the printRational() function with operator<<

Summary

- Today we covered
  - static members of a class
  - Operator overloading
- Completed pages 497 - 505, 547 - 555