Destructors, Get and Set, and Default Memberwise Assignment

Destructors

- The opposite of constructors
- Called whenever an object is destroyed
  - It is out of scope. For example, if it was a local variable in a function and the function has completed
- A destructor has no arguments and or return value
- Only one destructor allowed!
- No need for us to explicitly declare a destructor

Example

```
class Test
{
  private:
    int id;
  public:
    Test( int );
    ~Test();
};

Test::Test( int i )
{
  id = i;
  cout << "constructor for " << id << " is called\n";
}
Test::~Test()
{
  cout << "destructor for " << id << " is called\n";
}
```

What is the Output?

```
void funct();

int main()
{
  Test t1( 1 );
  funct();
  Test t3( 3 );

  return 0;
}

void funct()
{
  Test t2( 2 );
}
```

Set and Get Functions

- The principle of least privilege says that we should only provide outside members with access to data that is absolutely necessary
- Data members should therefore be set to private
- To modify and get access to that data, specific member functions need to be provided
- These are the Set and Get functions

Set and Get Functions

- The functions don’t need to be called set or get, but it has become commonplace to do this
- In the time class we could have the following set functions:
  - `void setTime( int, int, int );`
  - `void setHour( int );`
  - `void setMinute( int );`
  - `void setSecond( int );`

Get Functions

- For the Time class we would have the following get functions:

```
int getHour();
int getMinute();
int getSecond();
Time t4( 9, 25, 30 );
Time t5( 45, 90, 72 );
```

References to Private Data

- Although we may have declared the data inside of a class as private, there is a way to manipulate it directly (not use a member function)
- It is important that we are aware of this so that we can avoid it in the future

Example

```
class Test
{
  private:
    int id;
  public:
    int &setId( int );
    int getId();
};

int& Test::setId( int newId )
{
  id = ( newId >= 0 && newId <=10 )? newId : 0;
  return id;
}

int Test::getId()
{
  return id;
}
```

What is the Output?

```
int main()
{
  Test t1;

  int &testRef = t1.setId( 5 );

  cout << "Id is: " << t1.getId() << endl;

  testRef = 34;

  cout << "Id is: " << t1.getId() << endl;

  t1.setId( 4 ) = 52;

  cout << "Id is: " << t1.getId() << endl;

  return 0;
}
```

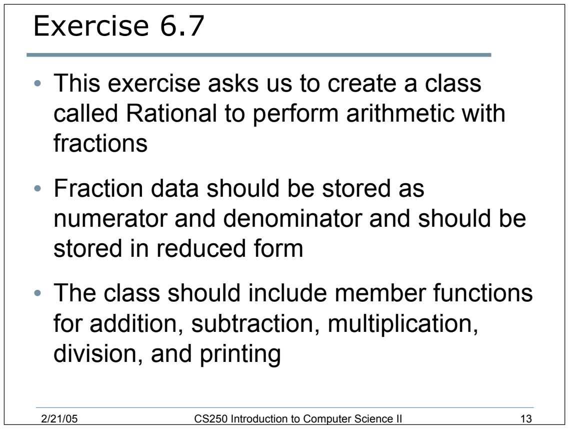
Default Memberwise Assignment

- It is possible to assign an object to another object of the same type
- This will assign every data member in the first object to the value of the equivalent data member in the second object

Example

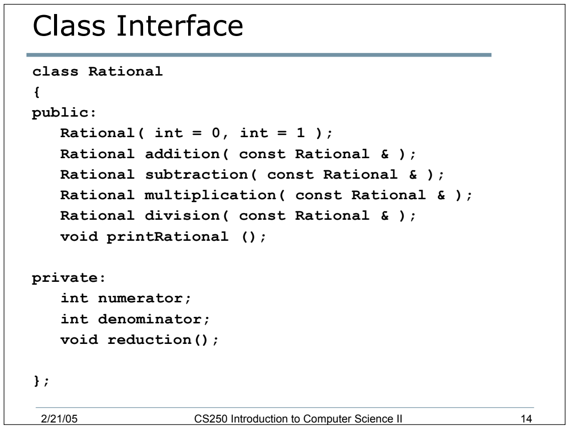
```
Time t1( 9, 25, 32 );
Time t2;

t2 = t1;

t2.printStandard();
```

- Let’s illustrate this further with another example



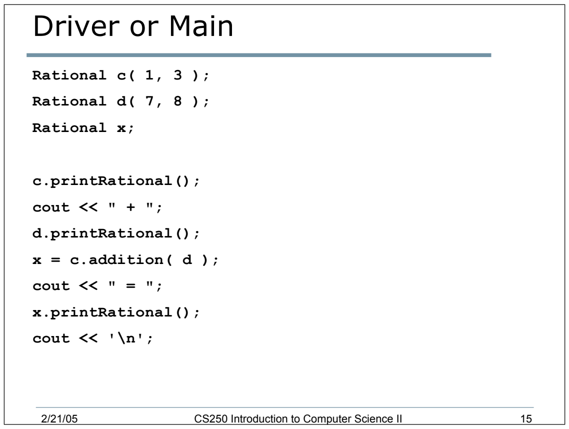
Exercise 6.7

- This exercise asks us to create a class called Rational to perform arithmetic with fractions
- Fraction data should be stored as numerator and denominator and should be stored in reduced form
- The class should include member functions for addition, subtraction, multiplication, division, and printing



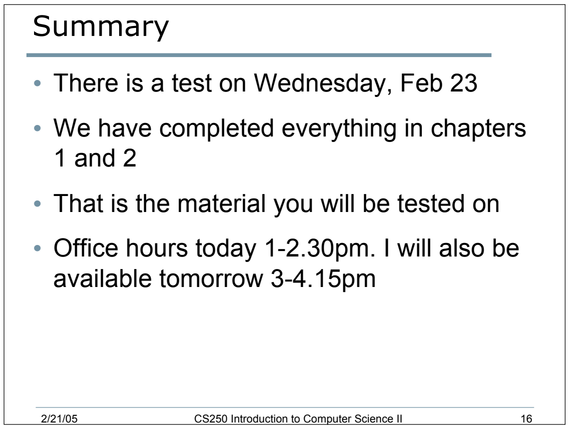
Class Interface

```
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();

};
```



Driver or Main

```
Rational c( 1, 3 );
Rational d( 7, 8 );
Rational x;

c.printRational();
cout << " + ";
d.printRational();
x = c.addition( d );
cout << " = ";
x.printRational();
cout << '\n';
```



Summary

- There is a test on Wednesday, Feb 23
- We have completed everything in chapters 1 and 2
- That is the material you will be tested on
- Office hours today 1-2.30pm. I will also be available tomorrow 3-4.15pm