Write a code segment that creates two unsigned integer, a and b, and a pointer, ptr that points to a. You should then change the value of a to 5 using ptr (and not a). Finally, set b to the value of a without using the pointer.

Consider the following class definition for the Rational class that we have been developing this quarter.

```
class Rational {
public:
  Rational(int numerator=0, int denomenator=1);
  Rational(const Rational& org);
  Rational& operator=(const Rational& rhs);
  ~Rational();
  void insert(std::ostream& os) const;
  Rational operator*(const Rational& rhs) const;
  Rational add(const Rational& rhs) const;
  Rational& operator*=(const Rational& rhs);
private:
  void reduce();
  int num;
  unsigned int den;
};
```

Implement the operator*= function (line 10).
Hint: The return statement should be: `return *this;`

In order to provide you with the best possible educational experience, I would like to get your comments on what is and isn't working in this class and lab.

At what moment were you most involved (excited, enthusiastic, . . .) in class/lab?

At what moment were you least involved (bored, disconnected, . . .) in class/lab?

What was the most helpful action taken by anyone in class/lab?

What was the most confusing action taken by anyone in class/lab?

What most surprised you?

What would be the first thing you would do differently if you were teaching the class?

Please add any additional comments you have. Indicate if you do *not* wish to have them appear in the summary returned to the class. Use the back of this page if necessary.

Suppose that the following two classes exist:

```
class Clock {
public:
   Clock();
   Clock(const Clock& orig);
   Clock(Time& now);
   ~Clock();
   Clock& operator=(Clock& rhs);
   void incrementMinutes();
   void decrementMinutes();
   void incrementHours();
   void decrementHours();
   void resetSeconds();
   void displayTime();
protected:
   ...

};
```

```
class AlarmClock : public Clock {
public:
   AlarmClock(const AlarmClock& orig);
   AlarmClock(Time tm = Time(),
              Time alm = Time(),
              bool on = false);
   ~AlarmClock();
   AlarmClock& operator=(AlarmClock& rhs);
   void incrementAlarmMinutes();
   void incrementAlarmHours();
   bool isAlarmOn() const;
   void alarmOn();
   void alarmOff();
private:
   ...

};
```

Identify which of the following lines of code are illegal.

```
Clock clk;
AlarmClock aclk;

clk.incrementMinutes();
clk.resetSeconds();
clk.displayTime();
clk.alarmOn();
clk.incrementAlarmHours();

aclk.incrementMinutes();
aclk.resetSeconds();
aclk.displayTime();
aclk.alarmOn();
aclk.incrementAlarmHours();

clk = aclk;
```

(7 points)

Found below is the Folder class we started last week.

```
class Folder : public DesktopItem {
public:
  Folder(std::string nm);
  Folder(const Folder& org);
  virtual ~Folder();
  Folder& operator=(const Folder& rhs);
  virtual DesktopItem* clone() const;
  virtual unsigned int size() const;
  bool add(DesktopItem* ptr);
  DesktopItem* find(const std::string& nm) const;
  bool remove(const std::string& nm);
protected:
  Folder();
  std::list<DesktopItem*> items;
};
```

Here is an implementation of the destructor for the class:

```
Folder::~Folder()
{
  std::list::iterator itr = items.begin();
  for( ; itr!=items.end(); ++itr) {
    delete *itr;
  }
}
```

Explain what is going on in the destructor.

(3 points)

Found below is the Folder class we started last week.

```
class Folder : public DesktopItem {
public:
  Folder(std::string nm);
  Folder(const Folder& org);
  virtual ~Folder();
  Folder& operator=(const Folder& rhs);
  virtual DesktopItem* clone() const;
  virtual unsigned int size() const;
  bool add(DesktopItem* ptr);
  DesktopItem* find(const std::string& nm) const;
  bool remove(const std::string& nm);
protected:
  Folder();
  std::list<DesktopItem*> items;
};
```

Implement the