```
/*** Program Name:   cis25Fall2011FinalMarcusLarsson.cpp
   * Written By:     Marcus Larsson
   * Date Written:   12/13/2011
   */

#include <iostream>
#include "MarcusBox.h"
#include "MarcusCylinder.h"
#include "MarcusSwap.h"
using namespace std;

void menu();
void printMenu();

void main() {

   menu();

   return;
}
void menu() {

   cout << "Starting at Line 50:" << endl; // Line 50
   MarcusFraction* fPtrA = new MarcusFraction(4, 1); // Line 100
   cout << *fPtrA << endl;

   MarcusFraction* fPtrC = new MarcusFraction(3, 1); // Line 300
   cout << *fPtrC << endl;

   MarcusFraction* fPtrE = new MarcusFraction(2, 1); // Line 500
   cout << *fPtrE << endl;

   MarcusFraction* fPtrF = new MarcusFraction(9, 1); // Line 600
   cout << *fPtrF << endl;

   cout << "\nStarting at Line 1880:" << endl; // Line 1880
   MarcusCircle* cPtrA = new MarcusCircle(*fPtrA); // Line 2000
   // cout << *cPtrA << endl;

   MarcusCircle* cPtrB = new MarcusCircle(*fPtrC); // Line 2100
   // cout << *cPtrB << endl;

   MarcusCircle* cPtrC = new MarcusCircle(*fPtrE); // Line 2200
   // cout << *cPtrC << endl;

   cout << "Starting at Line 2680:" << endl; // Line 2680
   MarcusCylinder* cyPtrA = new MarcusCylinder(*cPtrA, *fPtrA); // Line 3000
   // cout << *cyPtrA << endl;

   MarcusCylinder* cyPtrB = new MarcusCylinder(*cPtrB, *fPtrE); // Line 3100
   // cout << *cyPtrB << endl;

   MarcusCylinder* cyPtrC = new MarcusCylinder(*cPtrC, *fPtrC); // Line 3200
   // cout << *cyPtrC << endl;

   cout << "Starting at Line 3300:" << endl; // Line 3300
   MarcusRectangle* rPtrA = new MarcusRectangle(*fPtrA, *fPtrA); // Line 3400
   // cout << *rPtrA << endl;

   MarcusRectangle* rPtrB = new MarcusRectangle(*fPtrA, *fPtrE); // Line 3500
   // cout << *rPtrB << endl;

   MarcusRectangle* rPtrC = new MarcusRectangle(*fPtrA, *fPtrC); // Line 3600
   // cout << *rPtrC << endl;

   cout << "Starting at Line 3650:" << endl; // Line 3650
```

```
   MarcusBox* boxPtrA = new MarcusBox(*rPtrA, *fPtrA); // Line 3700
   // cout << *boxPtrA << endl;

   MarcusBox* boxPtrB = new MarcusBox(*rPtrB, *fPtrA); // Line 3800
   // cout << *boxPtrB << endl;

   MarcusBox* boxPtrC = new MarcusBox(*rPtrC, *fPtrC); // Line 3700
   // cout << *boxPtrC << endl;

   int option; // Line 3800
   // Didn't have time to set up dynamic arrays to store objects.
   option = 0;
   do {
      printMenu();
      cout << "\nSelect an option: ";
      cin >> option;
      switch(option) {
      case 1:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 2:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 3:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 4:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 5:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 6:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 7:
         cout << "\nNot implemented yet, sorry." << endl;
         break;
      case 8:
         cout << "\nGoodbye, CIS 25!." << endl; // Line 5000
         break;
      default:
         cout << "\nThat's not a menu option, try again." << endl; // Line 5100
         break;
      }
   } while (option != 8); // Line 5200
   return; // Line 10000
}
void printMenu() {
   cout << "\n*****************************************************************" << endl
           << "*                            MENU                               *" << endl
           << "*                                                               *" << endl
           << "* 1. Get & Display Largest Area (All Objects)                   *" << endl
           << "* 2. Get & Display Largest Volume (All Objects)                 *" << endl
           << "* 3. Display Areas from Largest to Smallest (All Circles)       *" << endl
           << "* 4. Display Areas from Largest to Smallest (All Rectangles)    *" << endl
           << "* 5. Display Volumes from Largest to Smallest (All Cylinders)   *" << endl
           << "* 6. Display Volumes from Largest to Smallest (All Boxes)       *" << endl
           << "* 7. Display Volumes from Largest to Smallest (All Objects)     *" << endl
           << "* 8. Quit                                                       *" << endl
           << "*                                                               *" << endl
           << "*****************************************************************" << endl;
   return;
}
```

```
/*** Header file or MarcusFraction Class
   * Program Name    cisMarcusFraction.h
   * Written By:     Marcus Larsson
   * Date Written:   11/09/2011
   */

#ifndef MARCUSFRACTION_H
#define MARCUSFRACTION_H

#include <iostream>
using namespace std;

class MarcusFraction {
public:
   // Constructors
   MarcusFraction();
   MarcusFraction(const MarcusFraction & arg);
   MarcusFraction(int arg);
   MarcusFraction(int arg1, int arg2);

   //Destructor
   ~MarcusFraction();

   // Accessors
   int getNum();
   int getDenom();

   // Mutators
   void setNum(int arg);
   void setDenom(int arg);
   void setFrac(int arg1, int arg2);
   void setFrac(int arg);

   // Declared friend functions
   friend MarcusFraction operator+(const MarcusFraction & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator-(const MarcusFraction & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator*(const MarcusFraction & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator/(const MarcusFraction & arg1, const MarcusFraction & arg2);

   // Operators using fraction for the first argument and an int for the second
   friend MarcusFraction operator+(const MarcusFraction & arg1, const int & arg2);
   friend MarcusFraction operator-(const MarcusFraction & arg1, const int & arg2);
   friend MarcusFraction operator*(const MarcusFraction & arg1, const int & arg2);
   friend MarcusFraction operator/(const MarcusFraction & arg1, const int & arg2);

   // Operators using an int for the first argument and a fraction for the second
   friend MarcusFraction operator+(const int & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator-(const int & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator*(const int & arg1, const MarcusFraction & arg2);
   friend MarcusFraction operator/(const int & arg1, const MarcusFraction & arg2);
   friend ostream& operator<<(ostream & arg1, const MarcusFraction & arg2);

   // Comparison operators
   friend bool operator<(const MarcusFraction & arg1, const MarcusFraction & arg2);
   friend int operator>(const MarcusFraction & arg1, const MarcusFraction & arg2);

   // Finding the GCD with Euclid's algorithm and reducing the MarcusFraction
   void reduce(int arg1, int arg2);

   // Other Member Functions
   void operator=(const MarcusFraction & arg);
   void operator=(const int & arg);

private:
   int iNum;
   int iDenom;
```

```
};

// Stand-alone functions
MarcusFraction operator+(const MarcusFraction & arg1, const MarcusFraction & arg2);
MarcusFraction operator-(const MarcusFraction & arg1, const MarcusFraction & arg2);
MarcusFraction operator*(const MarcusFraction & arg1, const MarcusFraction & arg2);
MarcusFraction operator/(const MarcusFraction & arg1, const MarcusFraction & arg2);
MarcusFraction operator+(const MarcusFraction & arg1, const int & arg2);
MarcusFraction operator-(const MarcusFraction & arg1, const int & arg2);
MarcusFraction operator*(const MarcusFraction & arg1, const int & arg2);
MarcusFraction operator/(const MarcusFraction & arg1, const int & arg2);
MarcusFraction operator+(const int & arg1, const MarcusFraction & arg2);
MarcusFraction operator-(const int & arg1, const MarcusFraction & arg2);
MarcusFraction operator*(const int & arg1, const MarcusFraction & arg2);
MarcusFraction operator/(const int & arg1, const MarcusFraction & arg2);
bool operator<(const MarcusFraction & arg1, const MarcusFraction & arg2);
int operator>(const MarcusFraction & arg1, const MarcusFraction & arg2);
ostream& operator<<(ostream & arg1, const MarcusFraction & arg2);

#endif
```

```
/*** Implementation file for the MarcusFraction class
   * Program Name:   cisMarcusFraction.cpp
   * Written By:     Marcus Larsson
   * Date Written:   11/09/2011
   */
#include <iostream>
#include "MarcusFraction.h"
using namespace std;
// Constructors
MarcusFraction::MarcusFraction() {
   iNum = 0;
   iDenom = 1;
}
MarcusFraction::MarcusFraction(const MarcusFraction & arg) {
   iNum = arg.iNum;
   iDenom = arg.iDenom;
}
MarcusFraction::MarcusFraction(int arg) {
   iNum = arg;
   iDenom = 1;
}
MarcusFraction::MarcusFraction(int arg1, int arg2) {
   iNum = arg1;
   if (arg2) {
      iDenom = arg2;
   } else {
      iDenom = 1;
      iNum = 0;
   }
   if (iDenom < 0) {
      iDenom *= -1;
      iNum *= -1;
   }
   reduce(arg1, arg2);
}
MarcusFraction::~MarcusFraction() {
   //cout << "MarcusFraction Destructor" << endl;
}
// Accessors
int MarcusFraction::getNum() {
   return iNum;
}
int MarcusFraction::getDenom() {
   return iDenom;
}
// Mutators
void MarcusFraction::setNum(int arg) {
   iNum = arg;
   return;
}
void MarcusFraction::setDenom(int arg) {
   if (arg) {
      iDenom = arg;
   } else {
      iDenom = 1;
   }
   if (iDenom < 0) {
      iDenom *= -1;
      iNum *= -1;
   }
}
void MarcusFraction::setFrac(int arg1, int arg2) {
   setNum(arg1);
   setDenom(arg2);
}
void MarcusFraction::setFrac(int arg) {
```

```
   setNum(arg);
   setDenom(1);
}
// Finding the GCD with Euclid's algorithm and reducing the MarcusFraction
void MarcusFraction::reduce(int arg1, int arg2) {
   int x;
   int y;
   int z;

   x = arg1;
   y = arg2;

   while ( y != 0 ) {
      z = x % y;
      x = y;
      y = z;
   }

   if (x < 0) {
      x *= -1;
   }

   iNum /= x;
   iDenom /= x;
return;
}
void MarcusFraction::operator=(const MarcusFraction & arg) {
   iNum = arg.iNum;
   iDenom = arg.iDenom;
   return;
}
void MarcusFraction::operator=(const int & arg) {
   iNum = arg;
   iDenom = 1;
}

// Stand-alone functions
MarcusFraction operator+(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   return MarcusFraction((arg1.iNum * arg2.iDenom) + (arg2.iNum * arg1.iDenom), arg1.iDenom * arg2.iDenom);
}
MarcusFraction operator-(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   return MarcusFraction((arg1.iNum * arg2.iDenom) - (arg2.iNum * arg1.iDenom), arg1.iDenom * arg2.iDenom);
}
MarcusFraction operator*(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   return MarcusFraction(arg1.iNum * arg2.iNum, arg1.iDenom * arg2.iDenom);
}
MarcusFraction operator/(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   return MarcusFraction(arg1.iNum * arg2.iDenom, arg1.iDenom * arg2.iNum);
}

MarcusFraction operator+(const MarcusFraction & arg1, const int & arg2) {
   return MarcusFraction((arg1.iNum) + (arg2 * arg1.iDenom), arg1.iDenom);
}
MarcusFraction operator-(const MarcusFraction & arg1, const int & arg2) {
   return MarcusFraction((arg1.iNum) - (arg2 * arg1.iDenom), arg1.iDenom);
}
MarcusFraction operator*(const MarcusFraction & arg1, const int & arg2) {
   return MarcusFraction(arg1.iNum * arg2, arg1.iDenom);
}
MarcusFraction operator/(const MarcusFraction & arg1, const int & arg2) {
   return MarcusFraction(arg1.iNum, arg1.iDenom * arg2);
}


MarcusFraction operator+(const int & arg1, const MarcusFraction & arg2) {
   return MarcusFraction((arg1 * arg2.iDenom) + (arg2.iNum), arg2.iDenom);
```

```
}
MarcusFraction operator-(const int & arg1, const MarcusFraction & arg2) {
   return MarcusFraction((arg1 * arg2.iDenom) - (arg2.iNum), arg2.iDenom);
}
MarcusFraction operator*(const int & arg1, const MarcusFraction & arg2) {
   return MarcusFraction(arg1 * arg2.iNum, arg2.iDenom);
}
MarcusFraction operator/(const int & arg1, const MarcusFraction & arg2) {
   return MarcusFraction(arg1 * arg2.iDenom, arg2.iNum);
}

bool operator<(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   int num1;
   int num2;
   bool temp;

   num1 = (arg1.iNum * arg2.iDenom);
   num2 = (arg2.iNum * arg1.iDenom);

   if (num1 < num2) {
      temp = 1;
   } else {
      temp = 0;
   }

   return temp;
}
int operator>(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   int num1;
   int num2;
   bool temp;

   num1 = (arg1.iNum * arg2.iDenom);
   num2 = (arg2.iNum * arg1.iDenom);

   if (num1 > num2) {
      temp = 1;
   } else {
      temp = 0;
   }

   return temp;
}

ostream& operator<<(ostream & arg1, const MarcusFraction & arg2) {
   if (arg2.iDenom == 1) {
      arg1 << arg2.iNum;
   } else {
      arg1 << arg2.iNum << "/" << arg2.iDenom;
   }
   return arg1;
}
```

```
/*** Header file or MarcusCircle Class
   * Program Name    MarcusCircle.h
   * Written By:     Marcus Larsson
   * Date Written:   11/29/2011
   */

#ifndef MARCUSCIRCLE_H
#define MARCUSCIRCLE_H

#include "MarcusPoint.h"

class MarcusCircle {
public:
   MarcusCircle();
   MarcusCircle(const MarcusCircle & arg);
   MarcusCircle(const MarcusPoint & arg1, const MarcusFraction & arg2);
   MarcusCircle(const MarcusFraction & arg);

   ~MarcusCircle();

   MarcusPoint getCenter() const ;
   MarcusFraction getRadius() const ;
   MarcusFraction getArea() const ;
   MarcusFraction getCircumference() const ;
   void setCenter(const MarcusPoint & arg);
   void setCenter(const MarcusFraction & arg1, const MarcusFraction & arg2);
   void setRadius(const MarcusFraction & arg);

   void print();

   void operator=(const MarcusCircle & arg);

   friend bool operator<(const MarcusCircle & arg1, const MarcusCircle & arg2);
   friend bool operator>(const MarcusCircle & arg1, const MarcusCircle & arg2);
   friend MarcusCircle operator+(const MarcusCircle & arg1, const MarcusCircle & arg2);


private:
   MarcusPoint center;
   MarcusFraction radius;
};

bool operator<(const MarcusCircle & arg1, const MarcusCircle & arg2);
bool operator>(const MarcusCircle & arg1, const MarcusCircle & arg2);
MarcusCircle operator+(const MarcusCircle & arg1, const MarcusCircle & arg2);

#endif
```

```
/* Implementation file for the MarcusCircle class
    Program Name: cisMarcusCircle.cpp
    Written By:      Marcus Larsson
    Date Written: 11/29/2011
   */

#include <iostream>
#include "MarcusCircle.h"
using namespace std;

MarcusCircle::MarcusCircle() {
   center;
   radius;
}
MarcusCircle::MarcusCircle(const MarcusCircle & arg) {
   center = arg.center;
   radius = arg.radius;
}
MarcusCircle::MarcusCircle(const MarcusPoint & arg1, const MarcusFraction & arg2) {
   center = arg1;
   radius = arg2;
}
MarcusCircle::MarcusCircle(const MarcusFraction & arg) {
   center.setX(arg);
   center.setY(arg);
   radius = arg;
}

MarcusCircle::~MarcusCircle() {
   //cout << "MarcusCircle Destructor." << endl;
}

MarcusPoint MarcusCircle::getCenter() const {
   return(center);
}
MarcusFraction MarcusCircle::getRadius() const {
   return(radius);
}
MarcusFraction MarcusCircle::getArea() const {
   MarcusFraction pi(157, 50);
   return MarcusFraction(pi * (radius * radius));
}
MarcusFraction MarcusCircle::getCircumference() const {
   MarcusFraction pi(157, 50);
   return MarcusFraction(2 * pi * radius);
}
void MarcusCircle::setCenter(const MarcusPoint & arg) {
   center = arg;
}
void MarcusCircle::setCenter(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   center.setX(arg1);
   center.setY(arg2);
}
void MarcusCircle::setRadius(const MarcusFraction & arg) {
   radius = arg;
}

void MarcusCircle::print() {
   cout << "Area: " << getArea() << endl
        << "Radius: " << radius << endl
        << "Center: " << center << endl
        << endl;
}

void MarcusCircle::operator=(const MarcusCircle & arg) {
   radius = arg.radius;
```

```
   center = arg.center;
}

bool operator<(const MarcusCircle & arg1, const MarcusCircle & arg2) {
   if (arg1.getArea() < arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
bool operator>(const MarcusCircle & arg1, const MarcusCircle & arg2) {
   if (arg1.getArea() > arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
MarcusCircle operator+(const MarcusCircle & arg1, const MarcusCircle & arg2) {
   MarcusCircle temp;
   temp.center = getMidPoint(arg1.center, arg2.center);
   temp.radius = arg1.radius + arg2.radius;
   return temp;
}
```

```
/*** Header file or MarcusCylinder Class
   * Program Name    MarcusCylinder.h
   * Written By:     Marcus Larsson
   * Date Written:   11/29/2011
   */

#ifndef MARCUSCYLINDER_H
#define MARCUSCYLINDER_H

#include "MarcusCircle.h"

class MarcusCylinder : public MarcusCircle {
public:
   MarcusCylinder();
   MarcusCylinder(const MarcusCylinder & arg);
   MarcusCylinder(const MarcusCircle & arg1, const MarcusFraction & arg2);

   ~MarcusCylinder();

   void setHeight(const MarcusFraction & arg);
   void setBase(const MarcusCircle & arg);
   MarcusFraction getHeight() const ;
   MarcusFraction getVolume() const ;
   MarcusFraction getSurfaceArea() const ;

   void print();

   void operator=(const MarcusCylinder & arg);

   friend bool operator<(const MarcusCylinder & arg1, const MarcusCylinder & arg2);
   friend bool operator>(const MarcusCylinder & arg1, const MarcusCylinder & arg2);
   friend MarcusCylinder operator+(const MarcusCylinder & arg1, const MarcusCylinder & arg2);



private:
   MarcusFraction height;
};

bool operator<(const MarcusCylinder & arg1, const MarcusCylinder & arg2);
bool operator>(const MarcusCylinder & arg1, const MarcusCylinder & arg2);
MarcusCylinder operator+(const MarcusCylinder & arg1, const MarcusCylinder & arg2);

#endif
```

```
/* Implementation file for the MarcusCylinder class
    Program Name: cisMarcusCylinder.cpp
    Written By:      Marcus Larsson
    Date Written: 11/29/2011
   */

#include <iostream>
#include "MarcusCylinder.h"
using namespace std;

MarcusCylinder::MarcusCylinder() {
   height;
   setRadius(0);
   setCenter(0, 0);
}
MarcusCylinder::MarcusCylinder(const MarcusCylinder & arg) {
   height = arg.height;
   setRadius(arg.getRadius());
   setCenter(arg.getCenter());
}
MarcusCylinder::MarcusCylinder(const MarcusCircle & arg1, const MarcusFraction & arg2) {
   height = arg2;
   setRadius(arg1.getRadius());
   setCenter(arg1.getCenter());
}
MarcusCylinder::~MarcusCylinder() {
   //cout << "MarcusCylinder Destructor." << endl;
}

void MarcusCylinder::setHeight(const MarcusFraction & arg) {
   height = arg;
}
void MarcusCylinder::setBase(const MarcusCircle & arg) {
   setRadius(arg.getRadius());
   setCenter(arg.getCenter());
}
MarcusFraction MarcusCylinder::getHeight() const {
   return(height);
}
MarcusFraction MarcusCylinder::getVolume() const {
   return (height * getArea());
}
MarcusFraction MarcusCylinder::getSurfaceArea() const {
   return((getCircumference() * height) + (getArea() * 2));
}

void MarcusCylinder::print(){
   cout << "Area: " << getSurfaceArea() << endl
        << "Volume: " << getVolume() << endl
        << "Base:  "  << endl
        << " - Radius: " << getRadius() << endl
        << " - Center: " << getCenter() << endl
        << endl;
}

void MarcusCylinder::operator=(const MarcusCylinder & arg) {
   height = arg.height;
   setRadius(arg.getRadius());
   setCenter(arg.getCenter());
}

bool operator<(const MarcusCylinder & arg1, const MarcusCylinder & arg2) {
   if (arg1.getVolume() < arg2.getVolume()) {
      return 1;
   } else {
      return 0;
```

```
   }
}
bool operator>(const MarcusCylinder & arg1, const MarcusCylinder & arg2) {
   if (arg1.getVolume() > arg2.getVolume()) {
      return 1;
   } else {
      return 0;
   }
}
MarcusCylinder operator+(const MarcusCylinder & arg1, const MarcusCylinder & arg2) {
   MarcusCylinder temp;
   // Setting the radius
   if (arg1.getArea() > arg2.getArea()) {
      temp.setRadius(arg1.getRadius());
   } else {
      temp.setRadius(arg2.getRadius());
   }
   // Setting the height
   if (arg1.height < arg2.height) {
      temp.setHeight(arg1.height);
   } else {
      temp.setHeight(arg2.height);
   }
   // Setting the center of the base circle
   temp.setCenter(getMidPoint(arg1.getCenter(), arg2.getCenter()));
   return temp;
}
```

```
/*** Header file or MarcusRectangle Class
   * Program Name    MarcusRecangle.h
   * Written By:     Marcus Larsson
   * Date Written:   11/29/2011
   */

#ifndef MARCUSRECTANGLE_H
#define MARCUSRECTANGLE_H

#include "MarcusPoint.h"

class MarcusRectangle {
public:
   MarcusRectangle();
   MarcusRectangle(const MarcusRectangle & arg);
   MarcusRectangle(const MarcusFraction & arg1, const MarcusFraction & arg2, const MarcusPoint & arg3);
   MarcusRectangle(const MarcusFraction & arg1, const MarcusFraction & arg2);

   ~MarcusRectangle();

   void setLength(const MarcusFraction & arg);
   void setWidth(const MarcusFraction & arg);
   void setLowerLeft(const MarcusPoint & arg);
   void setLowerLeft(const MarcusFraction & arg1, const MarcusFraction & arg2);
   MarcusFraction getLength() const ;
   MarcusFraction getWidth() const ;
   MarcusPoint getLowerLeft() const ;
   void print();

   friend bool operator<(const MarcusRectangle & arg1, const MarcusRectangle & arg2);
   friend bool operator>(const MarcusRectangle & arg1, const MarcusRectangle & arg2);

   MarcusFraction getArea() const ;
   MarcusFraction getPerimeter() const ;

   void operator=(const MarcusRectangle & arg);

private:
   MarcusPoint lowerLeft;
   MarcusFraction length;
   MarcusFraction width;
};

bool operator<(const MarcusRectangle & arg1, const MarcusRectangle & arg2);
bool operator>(const MarcusRectangle & arg1, const MarcusRectangle & arg2);

#endif
```

```
/* Implementation file for the MarcusRectangle class
    Program Name: cisMarcusRectangle.cpp
    Written By:      Marcus Larsson
    Date Written: 11/29/2011
   */

#include <iostream>
#include "MarcusRectangle.h"
using namespace std;

MarcusRectangle::MarcusRectangle() {
   length;
   width;
   lowerLeft;
}
MarcusRectangle::MarcusRectangle(const MarcusRectangle & arg) {
   length = arg.length;
   width = arg.width;
   lowerLeft = arg.lowerLeft;
}
MarcusRectangle::MarcusRectangle(const MarcusFraction & arg1, const MarcusFraction & arg2, const
   MarcusPoint & arg3) {
   length = arg1;
   width = arg2;
   lowerLeft = arg3;
}
MarcusRectangle::MarcusRectangle(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   length = arg1;
   width = arg2;
   lowerLeft.setX(arg1);
   lowerLeft.setY(arg2);
}

MarcusRectangle::~MarcusRectangle() {
   //cout << "MarcusRectangle Destructor." << endl;
}

void MarcusRectangle::setLength(const MarcusFraction & arg) {
   length = arg;
}
void MarcusRectangle::setWidth(const MarcusFraction & arg) {
   width = arg;
}
void MarcusRectangle::setLowerLeft(const MarcusPoint & arg) {
   lowerLeft = arg;
}
void MarcusRectangle::setLowerLeft(const MarcusFraction & arg1, const MarcusFraction & arg2) {
   lowerLeft.setPoint(arg1, arg2);
}
MarcusFraction MarcusRectangle::getLength() const {
   return length;
}
MarcusFraction MarcusRectangle::getWidth() const {
   return width;
}
MarcusPoint MarcusRectangle::getLowerLeft() const {
   return lowerLeft;
}

void MarcusRectangle::print() {
   cout << "Length: " << getLength() << endl
        << "Width: " << getWidth() << endl
        << "Corner: " << getLowerLeft() << endl
        << endl;
}
```

```
MarcusFraction MarcusRectangle::getArea() const {
   return(width * length);
}
MarcusFraction MarcusRectangle::getPerimeter() const {
   return((width * 2) + (length * 2));
}

void MarcusRectangle::operator=(const MarcusRectangle & arg) {
   width = arg.width;
   length = arg.length;
   lowerLeft = arg.lowerLeft;
}

bool operator<(const MarcusRectangle & arg1, const MarcusRectangle & arg2) {
   if (arg1.getArea() < arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
bool operator>(const MarcusRectangle & arg1, const MarcusRectangle & arg2) {
   if (arg1.getArea() > arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
```

```
/*** Header file or MarcusBox Class
   * Program Name    MarcusBox.h
   * Written By:     Marcus Larsson
   * Date Written:   11/29/2011
   */

#ifndef MARCUSBOX_H
#define MARCUSBOX_H

#include "MarcusRectangle.h"

class MarcusBox : public MarcusRectangle {
public:
   MarcusBox();
   MarcusBox(const MarcusBox & arg);
   MarcusBox(const MarcusRectangle & arg1, const MarcusFraction & arg2);

   ~MarcusBox();

   MarcusFraction getHeight() const ;
   void setHeight(const MarcusFraction & arg);
   void setBase(const MarcusRectangle & arg);

   MarcusFraction getVolume() const ;
   MarcusFraction getSurfaceArea() const ;

   friend bool operator<(const MarcusBox & arg1, const MarcusBox & arg2);
   friend bool operator>(const MarcusBox & arg1, const MarcusBox & arg2);

   void print();

   void operator=(const MarcusBox & arg);

private:
   MarcusFraction height;

};

bool operator<(const MarcusBox & arg1, const MarcusBox & arg2);
bool operator>(const MarcusBox & arg1, const MarcusBox & arg2);

#endif
```

```
/* Implementation file for the MarcusBox class
    Program Name: cisMarcusBox.cpp
    Written By:      Marcus Larsson
    Date Written: 11/29/2011
   */

#include <iostream>
#include "MarcusBox.h"
using namespace std;

MarcusBox::MarcusBox() {
   height;
   setLength(0);
   setWidth(0);
   setLowerLeft(0, 0);
}
MarcusBox::MarcusBox(const MarcusBox & arg) {
   height = arg.height;
   setLength(arg.getLength());
   setWidth(arg.getWidth());
   setLowerLeft(arg.getLowerLeft());
}
MarcusBox::MarcusBox(const MarcusRectangle & arg1, const MarcusFraction & arg2) {
   height = arg2;
   setLength(arg1.getLength());
   setWidth(arg1.getWidth());
   setLowerLeft(arg1.getLowerLeft());
}

MarcusBox::~MarcusBox() {
   //cout << "MarcusBox Destructor." << endl;
}

MarcusFraction MarcusBox::getHeight() const {
   return height;
}

void MarcusBox::setHeight(const MarcusFraction & arg) {
   height = arg;
}

void MarcusBox::setBase(const MarcusRectangle & arg) {
   setWidth(arg.getWidth());
   setLength(arg.getLength());
   setLowerLeft(arg.getLowerLeft());
}

MarcusFraction MarcusBox::getVolume() const {
   return(height * getLength() * getWidth());
}
MarcusFraction MarcusBox::getSurfaceArea() const {
   return((getArea() * 2) + ((height * getWidth()) * 2) + ((height * getLength()) * 2));
}

void MarcusBox::print() {
   cout << "Length: " << getLength() << endl
        << "Width: " << getWidth() << endl
        << "Height " << height << endl
        << "Corner: " << getLowerLeft() << endl
        << endl;
}

void MarcusBox::operator=(const MarcusBox & arg) {
   height = arg.height;
   setLength(arg.getLength());
   setWidth(arg.getWidth());
```

```
   setLowerLeft(arg.getLowerLeft());
}

bool operator<(const MarcusBox & arg1, const MarcusBox & arg2) {
   if (arg1.getArea() < arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
bool operator>(const MarcusBox & arg1, const MarcusBox & arg2) {
   if (arg1.getArea() > arg2.getArea()) {
      return 1;
   } else {
      return 0;
   }
}
```

```
/*** A swap-template used for finding largest and smallest values
   * Code borrowed from Charles Stuart
   */
#include <iostream>
using namespace std;

template <class T>
void swapLargest(T& arg1, T& arg2)
{
   T temp;
   if(arg1 < arg2)
   {
      temp = arg1;
      arg1 = arg2;
      arg2 = temp;
   }
   return;
};
template <class T>
void swapSmallest(T& arg1, T& arg2)
{
   T temp;
   if(arg1 > arg2)
   {
      temp = arg1;
      arg1 = arg2;
      arg2 = temp;
   }
   return;
};
```