```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/ShapeMain.cpp

#include <cstdlib> // EXIT_SUCCESS.
#include <cctype> // toupper.
#include "Utilities.hpp" // promptIntBetween, promptDoubleGE.
#include "AShape.hpp"
#include "Line.hpp"
#include "Rectangle.hpp"
#include "Circle.hpp"
#include "RightTriangle.hpp"
#include "MysteryShape.hpp"
#include "NullShape.hpp"
#include "ShapeMain.hpp"

int main() {
    const int NUM_SHAPES = 5;
    AShape *shapes[NUM_SHAPES];
    initialize(shapes, NUM_SHAPES);
    promptLoop(shapes, NUM_SHAPES);
    cleanUp(shapes, NUM_SHAPES);
    return EXIT_SUCCESS;
}

void initialize(AShape *shapes[], int cap) {
    for (int i = 0; i < cap; i++) {
        shapes[i] = new NullShape;
    }
}

void cleanUp(AShape *shapes[], int cap) {
    for (int i = 0; i < cap; i++) {
        delete shapes[i];
        shapes[i] = nullptr;
    };
}

void promptLoop(AShape *shapes[], int cap) {
    char response = '\0';
    do {
        cout << "\nThere are [0.." << cap - 1 << "] shapes." << endl;
        cout << "(m)ake  (c)lear  (a)rea  (p)erimeter  (s)cale  (d)isplay  (q)uit: ";
        cin >> response;
        switch (toupper(response)) {
        case 'M':
            makeShape(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'C':
            clearShape(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'A':
            printArea(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'P':
            printPerimeter(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'S':
            scaleShape(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'D':
            displayShape(shapes[promptIntBetween("Which shape?", 0, cap - 1)]);
            break;
        case 'Q':
```

```
            break;
        default:
            cout << "\nIllegal command." << endl;
            break;
        }
    } while (toupper(response) != 'Q');
}

void makeShape(AShape *&sh) {
    switch (shapeType()) {
    case 'L':
        delete sh;
        sh = new Line;
        break;
    case 'R':
        delete sh;
        sh = new Rectangle;
        break;
    case 'C':
        delete sh;
        sh = new Circle;
        break;
    case 'T':
        delete sh;
        sh = new RightTriangle;
        break;
     case 'M':
         delete sh;
         sh = new MysteryShape;
         break;
     default:
        break;
    }
    sh->promptAndSetDimensions();
}

char shapeType() {
    char ch;
    cout << "(l)ine  (r)ectangle  (c)ircle  right(t)riangle  (m)ystery: ";
    cin >> ch;
    ch = toupper(ch);
    while (ch != 'L' && ch != 'R' && ch != 'C' && ch != 'T' && ch != 'M') {
        cout << "Must be l, r, c, t, or m.  Which type? ";
        cin >> ch;
        ch = toupper(ch);
    }
    return ch;
}

void clearShape(AShape *&sh) {
    delete sh;
    sh = new NullShape;
}

void printArea(AShape *sh) {
    cout << "\nArea: " << sh->area() << endl;
}

void printPerimeter(AShape *sh) {
    cout << "\nPerimeter: " << sh->perimeter() << endl;
}

void scaleShape(AShape *sh) {
    sh->scale(promptDoubleGE("Scale factor?", 0.0));
}
```

```
void displayShape(AShape *sh) {
    cout << endl;
    sh->display(cout);
}


// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/Circle.cpp

#include "Circle.hpp"
#include "Utilities.hpp" // PI.

Circle::Circle(double radius) {
    if(radius < 0.0) {
        cerr << "Circle precondition violated: radius cannot be negative." << endl;
        throw -1;
    }
    _radius = radius;
}

double Circle::area() {
    return PI * _radius * _radius;
}

double Circle::perimeter() {
    return 2.0 * PI * _radius;
}

void Circle::scale(double factor) {
    _radius = _radius * factor;
}

void Circle::display(ostream &os) {
    os << "Circle\n" << "Radius: " << _radius << endl;
}

void Circle::promptAndSetDimensions() {
    _radius = promptDoubleGE("Radius?", 0.0);
}


// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/Line.cpp

#include "Line.hpp"
#include "Utilities.hpp"

Line::Line(double length) {
    if(length < 0.0) {
        cerr << "Line precondition violated: length cannot be negative." << endl;
        throw -1;
    }
    _length = length;
}

double Line::area() {
    return 0.0;
}

double Line::perimeter() {
    return _length;
}

void Line::scale(double factor) {
    _length = _length * factor;
}

void Line::display(ostream &os) {
    os << "Line\n" << "Length: " << _length << endl;
}

void Line::promptAndSetDimensions() {
    _length = promptDoubleGE("Length?", 0.0);
}


// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/MysteryShape.hpp

#ifndef MysteryShape_hpp
#define MysteryShape_hpp

#include "AShape.hpp"

class MysteryShape : public AShape {
private:
    double _side;

public:
   MysteryShape(double side = 0.0);
    // Pre: side >= 0.0
    // Post: This Mystery Shape is initialized with
    // side side.

    double area() override;
    double perimeter() override;
    void scale(double factor) override;
    void display(ostream &os) override;
    void promptAndSetDimensions() override;
};

#endif
// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/MysteryShape.cpp

#include "MysteryShape.hpp"
#include "Utilities.hpp"

MysteryShape::MysteryShape(double side) {
    if (side < 0.0) {
        cerr << "Mystery Shape precondition violated: side cannot be negative." << endl;
        throw -1;
    }
    _side = side;
}

double MysteryShape::area() {
    return _side * _side;
}

double MysteryShape::perimeter() {
    return 2.0 *(_side + _side);
}

void MysteryShape::scale(double factor) {
    _side = _side * factor;
}

void MysteryShape::display(ostream &os) {
    os << "Square" << endl;
    os << "Side: " << _side << endl;
}

void MysteryShape::promptAndSetDimensions() {
    _side = promptDoubleGE("Side?", 0.0);
}

// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/NullShape.cpp

#include "NullShape.hpp"

NullShape::NullShape() {
}

double NullShape::area() {
    return 0.0;
}

double NullShape::perimeter() {
    return 0.0;
}

void NullShape::scale(double factor) {
    factor * 0.0;
}

void NullShape::display(ostream &os) {
    os << endl;
}

void NullShape::promptAndSetDimensions() {
}


// new page
```

```
NullShape.cpp:21:12: warning: expression result unused [-Wunused-value]
    factor * 0.0;
    ~~~~~~ ^ ~~~
1 warning generated.
```

You should not have any compiler warnings

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/Rectangle.cpp

#include "Rectangle.hpp"
#include "Utilities.hpp"

Rectangle::Rectangle(double length, double width) {
    if (length < 0.0 || width < 0.0) {
        cerr << "Rectangle precondition violated: length and width cannot be negative."
<< endl;
        throw -1;
    }
    _length = length;
    _width = width;
}

double Rectangle::area() {
    return _length * _width;
}

double Rectangle::perimeter() {
    return 2.0 *(_length + _width);
}

void Rectangle::scale(double factor) {
    _length = _length * factor;
     _width = _width * factor;
}

void Rectangle::display(ostream &os) {
    os << "Rectangle" << endl;
    os << "Length: " << _length << endl;
    os << "Width: " << _width << endl;
}

void Rectangle::promptAndSetDimensions() {
    _length = promptDoubleGE("Length?", 0.0);
    _width = promptDoubleGE("Width?", 0.0);
}

// new page
```

```
// Olivia Lara
// Data Structures
// August 31, 2016

// File: Shape/RightTriangle.cpp

#include <cmath> // sqrt.
#include "RightTriangle.hpp"
#include "Utilities.hpp"

RightTriangle::RightTriangle(double base, double height) {
        if (base < 0.0 || height < 0.0) {
        cerr << "Right Triangle precondition violated: base and height cannot be
negative." << endl;
        throw -1;
    }
    _base = base;
    _height = height;
}

double RightTriangle::area() {
    return _base * _height * (.5);
}

double RightTriangle::perimeter() {
    double hyp;
    double x = (_base * _base)+ (_height + _height);
    hyp = sqrt(x);
    return hyp + _base + _height;
}

void RightTriangle::scale(double factor) {
     _base = _base * factor;
     _height = _height * factor;
}

void RightTriangle::display(ostream &os) {
    os << "Right Triangle" << endl;
    os << "Height: " << _height << endl;
    os << "Base: " << _base << endl;
}

void RightTriangle::promptAndSetDimensions() {
    _height = promptDoubleGE("Height?", 0.0);
    _base = promptDoubleGE("Base?", 0.0);
}

// new page
```

```
double RightTriangle::perimeter() {
    return _base + _height + sqrt(_base * _base + _height * _height);
}
```

```
Testing cs320-09 unit-scale
===========================

Line
Length: 6

Rectangle
Length: 9
Width: 12

Circle
Radius: 15

Tested cs320-09 unit-scale



Testing cs320-09 unit-right-triangle
====================================

Area: 6

Perimeter: 11.6904

Right Triangle
Height: 3
Base: 4

Right Triangle
Height: 9
Base: 12

Tested cs320-09 unit-right-triangle
```

-1