Lab 3 Solutions

2 Catch that bug

2.1

Line 20 is missing a semicolon. `doubleVal()` cannot be applied to `myPoint` because `myPoint` is `const`.

2.2

Line 11 contains an error because the function is declared `const`, i.e. as not modifying any instance variables, but it assigns `x` to another value.

2.3

`x` and `y` are private members and cannot be accessed outside of the class.

2.4

`setX` is missing the scope; the function should be declared as `void Point::setX(int newX) { x = newX; }`

2.5

Deleting a dynamically allocated array requires `delete[]`, not `delete`.

2.6

`p` is allocated using `new`, but is never deallocated with `delete`. Every piece of memory allocated with `new` must be deallocated somewhere with a corresponding `delete`.

3 Point

3.1 geometry.h

```
class Point {
    int x, y;

public:
    Point(int xx=0, int yy=0) {x = xx; y = yy;}
    int getX() const {return x;}
    int getY() const {return y;}
    void setX(const int xx) {x = xx;}
    void setY(const int yy) {y = yy;}
};
```

Note: the `getX` and `getY` functions should really have been declared as `const`, but we neglected to ask you to do this, so it's fine if you did not.

4 PointArray

4.1 geometry.h

```
class PointArray {
    int size;
    Point *points;

    void resize(int size);

public:
    PointArray();
    PointArray(const Point pts[], const int size);
    PointArray(const PointArray &pv);
    ~PointArray();

    void clear();
    int getSize() const { return size;}
    void push_back(const Point &p);
    void insert(const int pos, const Point &p);
    void remove(const int pos);
    Point *get(const int pos);
    const Point *get(const int pos) const;
};
```

4.2 geometry.cpp

```cpp
#include "geometry.h"

PointArray::PointArray() {
    size = 0;
    points = new Point[0]; // Allows deleting later
}

PointArray::PointArray(const Point ptsToCopy[], const int toCopySize
    ) {
    size = toCopySize;
    points = new Point[toCopySize];
    for(int i = 0; i < toCopySize; ++i) {
        points[i] = ptsToCopy[i];
    }
}

PointArray::PointArray(const PointArray &other) {
    // Any code in the PointArray class has access to
    // private variables like size and points
    size = other.size;
    points = new Point[size];
    for (int i = 0; i < size; i++) {
        points[i] = other.points[i];
    }
}

PointArray::~PointArray() {
    delete[] points;
}

void PointArray::resize(int newSize) {
    Point *pts = new Point[newSize];
    int minSize = (newSize > size ? size : newSize);
    for (int i = 0; i < minSize; i++)
        pts[i] = points[i];
    delete[] points;
    size = newSize;
    points = pts;
}

void PointArray::clear() {
    resize(0);
}
```

```

void PointArray::push_back(const Point &p) {
    resize(size + 1);
    points[size - 1] = p;
    // Could also just use insert(size, p);
}

void PointArray::insert(const int pos, const Point &p) {
    resize(size + 1);

    for (int i = size - 1; i > pos; i--) {
        points[i] = points[i-1];
    }

    points[pos] = p;
}

void PointArray::remove(const int pos) {
    if(pos >= 0 && pos < size) { // pos < size implies size > 0
        // Shift everything over to the left
        for(int i = pos; i < size - 2; i++) {
            points[i] = points[i + 1];
        }
        resize(size - 1);
    }
}

Point *PointArray::get(const int pos) {
    return pos >= 0 && pos < size ? points + pos : NULL;
}

const Point *PointArray::get(const int pos) const {
    return pos >= 0 && pos < size ? points + pos : NULL;
}
```

4.2.1

1. We need the `const` versions so that we can return read-only pointers for `const PointArray` objects. (If the `PointArray` object is read-only, we don't want to allow someone to modify a `Point` it contains just by using these functions.) However, many times we will have a non-`const PointArray` object, for which we may want to allow modifying the contained `Point` objects. If we had only `const` accessor functions, then even in such a case we would be returning a `const` pointer. To allow returning a non-`const` pointer in situations where we might want one, we need non-`const` versions of these

functions, as well.

5 Polygon and friends

5.1 Polygon

5.1.1 geometry.h

```cpp
class Polygon {
protected:
    static int numPolygons;
    PointArray points;

public:
    Polygon(const PointArray &pa);
    Polygon(const Point points[], const int numPoints);
    virtual double area() const = 0;
    static int getNumPolygons() {return numPolygons;}
    int getNumSides() const {return points.getSize();}
    const PointArray *getPoints() const {return &points;}
    ~Polygon() {--numPolygons;}
};
```

5.1.2 geometry.cpp

```cpp
int Polygon::n = 0;

Polygon::Polygon(const PointArray &pa) : points(pa) {
    ++numPolygons;
}

Polygon::Polygon(const Point pointArr[], const int numPoints) :
   points(pointArr, numPoints) {
    ++numPolygons;
}
```

5.2 Rectangle

5.2.1 geometry.h

```cpp
class Rectangle : public Polygon {
public:
    Rectangle(const Point &a, const Point &b);
    Rectangle(const int a, const int b, const int c, const int d);
    virtual double area() const;
};
```

5.2.2 geometry.cpp

```

Point constructorPoints[4];

Point *updateConstructorPoints(const Point &p1, const Point &p2,
    const Point &p3, const Point &p4 = Point(0,0)) {
    constructorPoints[0] = p1;
    constructorPoints[1] = p2;
    constructorPoints[2] = p3;
    constructorPoints[3] = p4;
    return constructorPoints;
}

Rectangle::Rectangle(const Point &ll, const Point &ur)
    : Polygon(updateConstructorPoints(ll, Point(ll.getX(), ur.getY()
        ),
                                       ur, Point(ur.getX(), ll.getY()
                                          )), 4) {}

Rectangle::Rectangle(const int llx, const int lly, const int urx,
    const int ury)
    : Polygon(updateConstructorPoints(Point(llx, lly), Point(llx,
        ury),
                                       Point(urx, ury), Point(urx,
                                          lly)), 4) {}

double Rectangle::area() const {
    int length = points.get(1)->getY() - points.get(0)->getY();
    int width  = points.get(2)->getX() - points.get(1)->getX();
    return std::abs((double)length * width);
}
```

(You'll need to add `#include <cmath>` at the top of your file to use the `abs` function.)

5.3 Triangle

5.3.1 geometry.h

```
class Triangle : public Polygon {
```

```
public:
    Triangle(const Point &a, const Point &b, const Point &c);
    virtual double area() const;
};
```

5.4 geometry.cpp

```
Triangle::Triangle(const Point &a, const Point &b, const Point &c)
    : Polygon(updateConstructorPoints(a, b, c), 3) {}

double Triangle::area() const {
    int dx01 = points.get(0)->getX() - points.get(1)->getX(),
        dx12 = points.get(1)->getX() - points.get(2)->getX(),
        dx20 = points.get(2)->getX() - points.get(0)->getX();
    int dy01 = points.get(0)->getY() - points.get(1)->getY(),
        dy12 = points.get(1)->getY() - points.get(2)->getY(),
        dy20 = points.get(2)->getY() - points.get(0)->getY();

    double a = std::sqrt(dx01*dx01 + dy01*dy01),
           b = std::sqrt(dx12*dx12 + dy12*dy12),
           c = std::sqrt(dx20*dx20 + dy20*dy20);

    double s = (a + b + c) / 2;

    return std::sqrt( s * (s-a) * (s-b) * (s-c) );
}
```

5.5 main.cpp

```
#include <iostream>
using namespace std;

#include "geometry.h"

void printAttributes(Polygon *p) {
    cout << "p's area is " << p->area() << ".\n";

    cout << "p's points are:\n";
    const PointArray *pa = p->getPoints();
    for(int i = 0; i < pa->getSize(); ++i) {
        cout << "(" << pa->get(i)->getX() << ", " << pa->get(i)->
            getY() << ")\n";
    }
}

int main(int argc, char *argv[]) {
    cout << "Enter lower left and upper right coords of rectangle as
        four space separated integers: ";
    int llx, lly, urx, ury;
    cin >> llx >> lly >> urx >> ury;
    Point ll(llx, lly), ur(urx, ury);
    Rectangle r(ll, ur);
    printAttributes(&r);

    cout << "Enter three coords of triangle as six space separated
       integers: ";
    int x1, y1, x2, y2, x3, y3;
    cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
    Point a(x1, y1), b(x2, y2), c(x3, y3);
    Triangle t(a, b, c);
    printAttributes(&t);

    return 0;
}
```

5.6 Questions

1. If the constructors were private, then we would not be able to create any `Point` objects.
2. When a `Polygon` is destroyed, the counter for number of `Polygons` created is decremented, and the `PointArray`'s destructor is implicitly called.
3. We had to make the fields of `Polygon` protected so that they could be accessed from `Rectangle` and `Triangle`, but not by arbitrary outside code.
4. The `getNumSides` from `Polygon` would be called, because the function is not `virtual`.

6 Strings

```
const string vowels = "aeiou";

string pigLatinify(const string s) {
    if(s.size() == 0) {
        // oops, empty string
        return s;
    }

    if(s.find("qu") == 0) { // Starts with "qu"
        return s.substr(2, s.size()-2) + "-" + s.substr(0, 2) + "ay";
    } else if(vowels.find(s[0]) != string::npos) {   // Starts with a vowel
        return s + "way";
    } else {
        return s.substr(1, s.size()-1) + "-" + s[0] + "ay";
    }
}
```