

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

```
1 class Point {
2     int x, y;
3
4 public:
5     Point(int xx=0, int yy=0) {x = xx; y = yy;}
6     int getX() const {return x;}
7     int getY() const {return y;}
8     void setX(const int xx) {x = xx;}
9     void setY(const int yy) {y = yy;}
10 };
```

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

```
1 class PointArray {
2     int size;
3     Point *points;
4
5     void resize(int size);
6
7 public:
8     PointArray();
9     PointArray(const Point pts[], const int size);
10    PointArray(const PointArray &pv);
11    ~PointArray();
12
13    void clear();
14    int getSize() const { return size;}
15    void push_back(const Point &p);
16    void insert(const int pos, const Point &p);
17    void remove(const int pos);
18    Point *get(const int pos);
19    const Point *get(const int pos) const;
20 };
```

4.2 geometry.cpp

```
1 #include "geometry.h"
2
3 PointArray::PointArray() {
4     size = 0;
5     points = new Point[0]; // Allows deleting later
6 }
7
8 PointArray::PointArray(const Point ptsToCopy[], const int toCopySize
9 ) {
10     size = toCopySize;
11     points = new Point[toCopySize];
12     for(int i = 0; i < toCopySize; ++i) {
13         points[i] = ptsToCopy[i];
14     }
15 }
16 PointArray::PointArray(const PointArray &other) {
17     // Any code in the PointArray class has access to
18     // private variables like size and points
19     size = other.size;
20     points = new Point[size];
21     for (int i = 0; i < size; i++) {
22         points[i] = other.points[i];
23     }
24 }
25
26 PointArray::~PointArray() {
27     delete[] points;
28 }
29
30 void PointArray::resize(int newSize) {
31     Point *pts = new Point[newSize];
32     int minSize = (newSize > size ? size : newSize);
33     for (int i = 0; i < minSize; i++)
34         pts[i] = points[i];
35     delete[] points;
36     size = newSize;
37     points = pts;
38 }
39
40 void PointArray::clear() {
41     resize(0);
42 }
```

```

43
44 void PointArray::push_back(const Point &p) {
45     resize(size + 1);
46     points[size - 1] = p;
47     // Could also just use insert(size, p);
48 }
49
50 void PointArray::insert(const int pos, const Point &p) {
51     resize(size + 1);
52
53     for (int i = size - 1; i > pos; i--) {
54         points[i] = points[i-1];
55     }
56
57     points[pos] = p;
58 }
59
60 void PointArray::remove(const int pos) {
61     if(pos >= 0 && pos < size) { // pos < size implies size > 0
62         // Shift everything over to the left
63         for(int i = pos; i < size - 1; i++) {
64             points[i] = points[i + 1];
65         }
66         resize(size - 1);
67     }
68 }
69
70 Point *PointArray::get(const int pos) {
71     return pos >= 0 && pos < size ? points + pos : NULL;
72 }
73
74 const Point *PointArray::get(const int pos) const {
75     return pos >= 0 && pos < size ? points + pos : NULL;
76 }

```

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

```
1 class Polygon {
2 protected:
3     static int numPolygons;
4     PointArray points;
5
6 public:
7     Polygon(const PointArray &pa);
8     Polygon(const Point points[], const int numPoints);
9     virtual double area() const = 0;
10    static int getNumPolygons() {return numPolygons;}
11    int getNumSides() const {return points.getSize();}
12    const PointArray *getPoints() const {return &points;}
13    ~Polygon() {--numPolygons;}
14 };
```

5.1.2 geometry.cpp

```
1 int Polygon::n = 0;
2
3 Polygon::Polygon(const PointArray &pa) : points(pa) {
4     ++numPolygons;
5 }
6
7 Polygon::Polygon(const Point pointArr[], const int numPoints) :
8     points(pointArr, numPoints) {
9     ++numPolygons;
10 }
```

5.2 Rectangle

5.2.1 geometry.h

```
1 class Rectangle : public Polygon {
2 public:
3     Rectangle(const Point &a, const Point &b);
```

```

4   Rectangle(const int a, const int b, const int c, const int d);
5   virtual double area() const;
6 };

```

5.2.2 geometry.cpp

```

1
2 Point constructorPoints[4];
3
4 Point *updateConstructorPoints(const Point &p1, const Point &p2,
    const Point &p3, const Point &p4 = Point(0,0)) {
5     constructorPoints[0] = p1;
6     constructorPoints[1] = p2;
7     constructorPoints[2] = p3;
8     constructorPoints[3] = p4;
9     return constructorPoints;
10 }
11
12 Rectangle::Rectangle(const Point &ll, const Point &ur)
13     : Polygon(updateConstructorPoints(ll, Point(ll.getX(), ur.getY()
14         ),
15         ur, Point(ur.getX(), ll.getY()
16         )), 4) {}
17
18 Rectangle::Rectangle(const int llx, const int lly, const int urx,
19     const int ury)
20     : Polygon(updateConstructorPoints(Point(llx, lly), Point(llx,
21         ury),
22         Point(urx, ury), Point(urx,
23         lly)), 4) {}
24 }
25
26 double Rectangle::area() const {
27     int length = points.get(1)->getY() - points.get(0)->getY();
28     int width  = points.get(2)->getX() - points.get(1)->getX();
29     return std::abs((double)length * width);
30 }

```

(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

```

1 class Triangle : public Polygon {

```

```

2 public:
3     Triangle(const Point &a, const Point &b, const Point &c);
4     virtual double area() const;
5 };

```

5.4 geometry.cpp

```

1 Triangle::Triangle(const Point &a, const Point &b, const Point &c)
2     : Polygon(updateConstructorPoints(a, b, c), 3) {}
3
4 double Triangle::area() const {
5     int dx01 = points.get(0)->getX() - points.get(1)->getX(),
6         dx12 = points.get(1)->getX() - points.get(2)->getX(),
7         dx20 = points.get(2)->getX() - points.get(0)->getX();
8     int dy01 = points.get(0)->getY() - points.get(1)->getY(),
9         dy12 = points.get(1)->getY() - points.get(2)->getY(),
10        dy20 = points.get(2)->getY() - points.get(0)->getY();
11
12    double a = std::sqrt(dx01*dx01 + dy01*dy01),
13           b = std::sqrt(dx12*dx12 + dy12*dy12),
14           c = std::sqrt(dx20*dx20 + dy20*dy20);
15
16    double s = (a + b + c) / 2;
17
18    return std::sqrt( s * (s-a) * (s-b) * (s-c) );
19 }

```

5.5 main.cpp

```

1 #include <iostream>
2 using namespace std;
3
4 #include "geometry.h"
5
6 void printAttributes(Polygon *p) {
7     cout << "p's area is " << p->area() << ".\n";
8
9     cout << "p's points are:\n";
10    const PointArray *pa = p->getPoints();
11    for(int i = 0; i < pa->getSize(); ++i) {
12        cout << "(" << pa->get(i)->getX() << ", " << pa->get(i)->
13            getY() << ")\n";
14    }
15 }

```

```

14 }
15
16 int main(int argc, char *argv[]) {
17     cout << "Enter lower left and upper right coords of rectangle as
         four space separated integers: ";
18     int llx, lly, urx, ury;
19     cin >> llx >> lly >> urx >> ury;
20     Point ll(llx, lly), ur(urx, ury);
21     Rectangle r(ll, ur);
22     printAttributes(&r);
23
24     cout << "Enter three coords of triangle as six space separated
         integers: ";
25     int x1, y1, x2, y2, x3, y3;
26     cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
27     Point a(x1, y1), b(x2, y2), c(x3, y3);
28     Triangle t(a, b, c);
29     printAttributes(&t);
30
31     return 0;
32 }

```

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

```

1 const string vowels = "aeiou";
2
3 string pigLatinify(const string s) {
4     if(s.size() == 0) {
5         // oops, empty string
6         return s;
7     }
8

```



```
9     if(s.find("qu") == 0) { // Starts with "qu"
10         return s.substr(2, s.size()-2) + "-" + s.substr(0, 2) + "ay"
11         ;
12     } else if(vowels.find(s[0]) != string::npos) { // Starts with
13         a vowel
14         return s + "way";
15     } else {
16         return s.substr(1, s.size()-1) + "-" + s[0] + "ay";
17     }
```

MIT OpenCourseWare

<http://ocw.mit.edu>

6.096 Introduction to C++

January (IAP) 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.