CS180 - Classes & Variable Types

Note Title 1/26/2011

Announcements

- HW due today
- Tutoring is now open for business
- Next assignment up soon
- Nice talk today at Math/CS club
  Cryptography
  4pm in Ritter Lobby

Defining a class: Remember the Point class?

```
class Point {
  private:
    double _x;                          // explicit declaration of data members
    double _y;

  public:
    Point( ) : _x(0), _y(0) { }         // constructor

    double getX( ) const {              // accessor
      return _x;
    }

    void setX(double val) {             // mutator
      _x = val;
    }

    double getY( ) const {              // accessor
      return _y;
    }

    void setY(double val) {             // mutator
      _y = val;
    }

};                                      // end of Point class (semicolon is required)
```

data in the class

Point mypoint;

cout << mypoint._x << endl;

ERROR

mypoint.getX()

Classes - differences:

1. Data (public or private) is explicitly declared, not just used in constructor.

2. Constructor!
   - name is always same as class
   - no return type
   - can initialize variables in a list (initialization list)

```
Point() : _x(0), _y(0)

Point(int x=0, int y=0) : _x(x), _y(y) {}

Point() {
  _x = 0;
  _y = 0; }
```

A more complicated constructor:

input parameters

```
Point(double initialX=0.0, double initialY=0.0) : _x(initialX), _y(initialY) { }
```

- Allows default parameters, but body is still empty.

Other things to note:

③ No self! Can just use _x or _y, & understood to be attributes of current object.

(Could use this, ie this._x, if necessary.)

④ Access control – public versus private
- main can't touch private variables!
- functions are often public

→ can't make local variables w/same name:
Ex: no int _x; (inside Point function)

Other things to note (cont):

⑤ accessor versus mutator:

_x = 6; ← compiler error

```
double getX( ) const {          // accessor
  return _x;
}
```

forces it to not change data

difference?

```
void setX(double val) {         // mutator
  _x = val;
}
```

Forced by compiler:

in main:

→ mypoint.getX() = 5;   error

Robust Point class cont:

might add some functionality:

```
double distance(Point other) const {
  double dx = _x − other._x;
  double dy = _y − other._y;
  return sqrt(dx * dx + dy * dy);        // sqrt imported from cmath library
}

void normalize( ) {
  double mag = distance( Point( ) );      // measure distance to the origin
  if (mag > 0)
    scale(1/mag);
}

Point operator+(Point other) const {
  return Point(_x + other._x, _y + other._y);
}

Point operator*(double factor) const {
  return Point(_x * factor, _y * factor);
}

double operator*(Point other) const {
  return _x * other._x + _y * other._y;
}
};   // end of Point class (semicolon is required)
```

← can access private data if inside class

← mypoint.distance(other)

newpt = mypoint + otherpt

Things to note:

1) _x + other._x ⟸ allowed <u>if</u> inside the class

2) using operator+, will be x + y (note const)

$z = x + y$

$x = x + y$

3) two versions of *

in some instances, have more than 1 interpretation:

$2 * (3, 6) = (6, 12)$

$(3, 6) * 2$

$(5, 2) * (1, 3) = 5 \cdot 1 + 2 \cdot 3 = 11$

Additional functions (Not in class)

must be in class

x + y

```
// Free-standing operator definitions, outside the formal Point class definition
Point operator*(double factor, Point p) {
   return p * factor;                        // invoke existing form with Point as left operand
}

ostream& operator<<(ostream& out, Point p) {
   out << "<" << p.getX( ) << "," << p.getY( ) << ">";      // display using form <x,y>
   return out;
}
```

2 * (3, 6)

cout << mypoint << otherpoint;

<3, 6>

Why outside of class?

C++ does not allow right operator to be instance of an object.

Inheritance

What is it?

A way to create a class that can steal another class' functions

(a way to be lazy)

Example: Square class

#include Rectangle.cpp h

```
class Square : public Rectangle {
  public:
    Square(double size=10, Point center=Point( )) :
      Rectangle(size, size, center)        // parent constructor
      { }

    void setHeight(double h) { setSize(h); }
    void setWidth(double w) { setSize(w); }

    void setSize(double size) {
      Rectangle::setWidth(size);       // make sure to invoke PARENT version
      Rectangle::setHeight(size);      // make sure to invoke PARENT version
    }

    double getSize( ) const { return getWidth( ); }
};   // end of Square
```

↑ scope to parent class' function

Can't say _height = 5

Other issues:

A new type of data:
- We have seen public & private. Public is inherited and private is not.

But what about data which should be private, but also should be inherited?

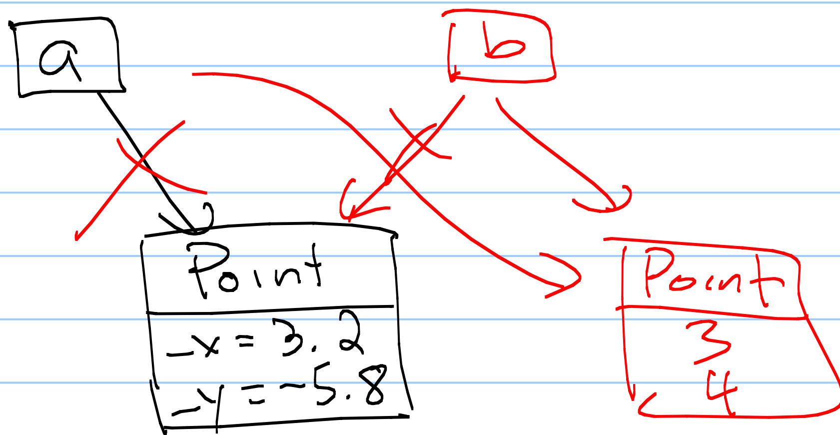
Ex: ~~public:~~
~~int height;~~
~~int width;~~

protected:
int _height;
int _width;

allows any child classe to gain access

Objects & Memory Management

In Python, variables were pointers to data:



b = a;
b = Point(3, 4);

a = b;

this gets erased (automatic garbage collection)

Example: Square class

```
class Square : public Rectangle {
  public:
    Square(double size=10, Point center=Point( )) :
      Rectangle(size, size, center)        // parent constructor
      { }

    void setHeight(double h) { setSize(h); }
    void setWidth(double w) { setSize(w); }

    void setSize(double size) {
      Rectangle::setWidth(size);       // make sure to invoke PARENT version
      Rectangle::setHeight(size);      // make sure to invoke PARENT version
    }

    double getSize( ) const { return getWidth( ); }
};  // end of Square
```

Other issues:

A new type of data:

- We have seen public & private. Public is inherited and private is not.

But what about data which should be private, but also should be inherited?

<u>Ex</u>: public:

```
int height;
int width;
```

Speeding up the Point class:

original:

```
double distance(Point other) const {
```

faster:

```
double distance(const Point& other) const {
```

Another:

```
Point operator+(const Point& other) const {
  return Point(_x + other._x, _y + other._y);
}
```

Note: Return type is still value. Why?