Announcements

- HW1 due Sat.
- Look for HW2 on website soon.
- Lab tomorrow (already posted).
also: global variables

A note on variable scopes:

```cpp
int main () {
  int a;
  cin >> a;
  if (a > 0)
    int b = 12;
  else
    int b = 16;
  // b is destroyed
  cout << "a is " << a << endl;
  cout << "b is " << b << endl;  // undefined variable
  // a is destroyed
```
Classes

What is a class? Very useful!
- contain functions
- collection of related data
Creating an instance of a class

Example:

```c++
string s;  // calling default constructor
string greeting("Hello");  // has input param for constructor
```

Never:

```c++
string s();
```

Why? Empty function called `s`

Never:

```c++
string("Hello") greeting;
```

Why? Compile error
class Point {
private:
    double x;
    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;
    }

} // explicit declaration of data members
Classes:

1. Data — public or private — is explicitly declared, not just used in constructor.
   This is done inside the class, but not inside a function.

Why?

Scope would end in function.
2 Constructor Function
- name: always same as class
- no return type
- can initialize variables in a list

\[ \text{Point( ) : } x(0), y(0) \{ \} \]

\[ \text{Point( ) : } x(0), y(0) \{ \} \rightarrow \text{Point( ) : } x(0), y(0) \{ \} \]

\[ \text{Point(new initialX=0.0, new initialY=0.0) : } x(\text{initialX}), y(\text{initialY}) \{ \} \]
Other differences

3. No self! Can just use -x or -y and it immediately scopes to the class attributes.

(There is a "this", but its usage is a bit more complex.)

4. Access control—public versus private. Compiler forces this:

functions and data must be set as public or private
Accessor versus mutator:

```cpp
double getX() const { return x; }
void setX(double val) { x = val; }
```

changes class date

accessor function

- compiler enforces it

const double grav = 9.81;
Robust point class: add 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)

(3,4) + (2,2)
p1 + p2
```
Important things

1) \( x + \text{others} \cdot x \) \( \leftarrow \text{allowed only inside the class} \)

2) using `operator+` : can say `pt1.operator+(pt2)`;
   Point `newpt = pt1 + pt2`;

3) two versions of `*` in Python; one function: `used .isInstance`
   `pt = pt * 2`; \( \leftarrow \text{pt} . \text{operator*}(2) \)
   `pt = pt1 * pt2`;
Additional functions

3: // end of Point class
(Not in the class)
2 * p^+

// 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;
}

Why?

cout << pt1 << pt2;

cout << "my point is" << pt1;
Inheritance

What is inheritance?

A way to be lazy:

Child class which steals the data and functions of parent class.

Person

   Student   Professor
Example: Square class

class Square : public Rectangle {
public:
    Square(double size=10, Point center=Point(0)) :
        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( ); } // go to parent's version automatically
}; // end of Square
Other issues

A new type of data. So far, have seen public and private.

Anyone can see. No one can see.

What about data that main can't have but child classes should?

protected:

// variables or functions
Objects

In Python, variables are pointers to actual data.

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

```
<table>
<thead>
<tr>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 3</td>
</tr>
<tr>
<td>y = 4</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 3.2</td>
</tr>
<tr>
<td>y = 5.8</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
</table>
```

```
<table>
<thead>
<tr>
<th>b</th>
</tr>
</thead>
</table>
```
C++: More versatile

C++ allows for 3 different types of variables:

1. Value
2. Reference: 
3. Pointer: 

When a variable is created, a precise amount of memory is set aside.

Point a:
Point b(5,7):

<table>
<thead>
<tr>
<th>a : Point</th>
<th>b : Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 0.0</td>
<td>x = 5.0</td>
</tr>
<tr>
<td>y = 0.0</td>
<td>y = 7.0</td>
</tr>
</tbody>
</table>

More efficient (for both speed & space).
Now set \( a = b \):

<table>
<thead>
<tr>
<th>a : Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 5.0 )</td>
</tr>
<tr>
<td>( y = 7.0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b : Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = 5.0 )</td>
</tr>
<tr>
<td>( y = 7.0 )</td>
</tr>
</tbody>
</table>

depth copy

They stay separate!

Everything is a deep copy!
Functions: passing by value

```c++
bool isOrigin(Point pt) {
    return pt.getX() == 0 && pt.getY() == 0;
}
```

When someone calls `isOrigin(myPoint)`, the value of `pt` is initialized as a new, separate variable.

Essentially, the line:

```
Point pt (myPoint);
```

is run at the beginning of the function!

So do changes to the point last?

**No**