Objects and classes

Victor Eijkhout and Carrie Arnold and Charlie Dey

Fall 2017

Classes

Classes look a bit like structures

```
class Vector {
public:
  double x,y;
};

int main() {
  Vector p1;
  p1.x = 1.; p1.y = 2.;
```

We'll get to that 'public' in a minute.

Class initialization and use

Use a *constructor*:

```
class Vector {
private:
  double x,y;
public:
  Vector( double userx,double usery ) {
    x = userx; y = usery;
  }
  /* ... */
};
  /* ... */
  Vector p1(1.,2.);
```

Member initialization

Other syntax for initialization:

```
class Vector {
private:
  double x,y;
public:
  Vector( double userx,double usery ) : x(userx),y(usery) {
  }
  /* ... */
};
```

Private data

```
class Vector {
private:
  double vx,vy;
public:
  Vector( double x,double y ) {
    vx = x; vy = y;
  };
  double x() { return vx; }; // 'accessor'
  double y() { return vy; };
```

Accessor for setting private data

```
void setx( double newx ) { vx = newx; };
void sety( double newy ) { vy = newy; };
/* ... */
p1.setx(3.12);
/* ILLEGAL: p1.x() = 5; */
```

Methods

Functions on objects

```
class Vector {
private:
  double vx,vy;
public:
  Vector( double x,double y ) {
    vx = x; vy = y;
  };
  double length() { return sqrt(vx*vx + vy*vy); };
  double angle() { return 0.; /* something trig */; };
};

int main() {
  Vector p1(1.,2.);
  cout << "p1 has length " << p1.length() << endl;
```

We call such internal functions 'methods'

Methods that alter the object

```
class Vector {
  /* ... */
  void scaleby( double a ) {
    vx *= a; vy *= a; };
  /* ... */
};
  /* ... */
  Vector p1(1.,2.);
  cout << "p1 has length " << p1.length() << endl;
  p1.scaleby(2.);
  cout << "p1 has length " << p1.length() << endl;
```

Methods that create a new object

```
class Vector {
  /* ... */
  Vector scale( double a ) {
    return Vector( vx*a, vy*a ); };
  /* ... */
};
  /* ... */
  cout << "p1 has length " << p1.length() << endl;
  Vector p2 = p1.scale(2.);
  cout << "p2 has length " << p2.length() << endl;
```

Default constructor

```
Vector p1(1.,2.), p2;
cout << "p1 has length " << p1.length() << endl;
p2 = p1.scale(2.);
cout << "p2 has length " << p2.length() << endl;
```

gives (g++; different for intel):

```
pointdefault.cxx: In function 'int main()':
pointdefault.cxx:32:21: error: no matching function for call to
                  'Vector::Vector()'
   Vector p1(1.,2.), p2;
```

So:

```
Vector() {};
Vector( double x,double y ) {
  vx = x; vy = y;
};
```

Exercise 1

Make class `Point` with a constructor

```
Point( float xcoordinate, float ycoordinate );
```

Write a method `distance_to_origin` that returns a `float`.

Write a method `printout` that uses `cout` to display the point.

Write a function `distance` so that if `p,q` are `Point` objects,

```
p.distance(q)
```

computes the distance.

Exercise 2

Make a class `LinearFunction` with a constructors:

```
LinearFunction( Point input_p1,Point input_p2 );
```

and a function

```
float evaluate_at( float x );
```

which you can use as:

```
LinearFunction line(p1,p2);
cout << "Value at 4.0: " << line.evaluate_at(4.0) << endl;
```

Exercise 3

Make a class `LinearFunction` with two constructors:

```
LinearFunction( Point input_p2 );
LinearFunction( Point input_p1,Point input_p2 );
```

where the first stands for a line through the origin.
Implement again the `evaluate` function so that

```
LinearFunction line(p1,p2);
cout << "Value at 4.0: " << line.evaluate_at(4.0) << endl;
```