Objects and classes

Victor Eijkhout and Charlie Dey

spring 2017

Classes

Classes look a bit like objects

```
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

Use a *constructor*:

```
class Vector {
public:
  double x,y;
  Vector( double userx,double usery ) {
    x = userx; y = usery;
  }
};

int main() {
  Vector p1(1.,2.);
```

Member initialization

Other syntax for initialization:

```
class Vector {
public:
  double x,y;
  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; };
  double y() { return vy; };
};

int main() {
  Vector p1(1.,2.);
  cout << "p1 = " << p1.x() << "," << p1.y() << endl;
```

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); };
};

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;
```

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:

```
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 );
```

and a function `distance` so that if `p,q` are `Point` objects, the call

```
  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;
```