CS180 - Classes in C++

Note Title 9/8/2011

Announcements

- HW1 due Sat.
- Look for HW2 on website soon.
- Lab tomorrow (posted on lab page)
  - don't forget to email prelab before class!!

A note on variable scopes:

how long it exists

```
int main () {
    int a;
    if (a > 0) {
        int b = 12; } // b is destroyed
    else {
        int b = 16; } // b is destroyed

    cout << "a is " << a << endl;
    cout << "b is " << b << endl; ← compiler error
} // a is destroyed
```

for loops:

```
for (int i=0; i<val; i++) {


} // i is destroyed
  cout << i;   <-- error
```

Arrays as inputs to functions

Example: Write a function to specify if sum of values in an array is even.

```
bool evenSum(int anArray[], int n) {
    int sum = 0;
    for (int i=0; i<n; i++)
        sum = sum + anArray[i];

    return ((sum %2) == 0);
}
```

Note:

- int anArray[] actually makes a (the array) a pointer!
  (More on these later...)

Doesn't copy whole array but can pretend that it does - just use it like an array.

To call:

```
int main() {
// create & put values in myArray
  if (evenSum(myArray, length))
    cout << "The sum is even" << endl;
}
```

You do this

Classes

What is a class?

data field: a collection of data

store arbitrary collections of data

along with allowed operations

Ex: records for people

Creating an instance of a class

Example:

```
string s;
string greeting("Hello");
```

← calls Constructor for class

↑ input to initialize

Never:

```
string s( );
```

Why? declares a function named s with no inputs which does nothing

Never:

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

Why? compiler hates it

Example:

```
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;
  }
};
```

name of class

accessible only inside class

declaration list

int x;

Constructor

no semicolon

no self._x

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 only be that function.

② Constructor Function

- name: same as class

- no return type (only one!)

- can initialize variables via a list

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

⟺

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

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

Other differences

(3) No self! Can just use _x or _y, & 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.

```
Point mypoint;
mypoint._x = 2;  <-> compiler error
```

⑤ Accessor versus mutator

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

can enforce this in C++.

Robust Point class : add functionality

in main:

```
double dist =
  pt1.distance(pt2);
```

```
Point p3 = pt1 + pt2;
   or
p3 = pt1.operator+(pt2);
```

```
pt1 * pt2;
pt1 * 3.0;
```

```
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);
}

--add--
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)
```

Important things

1) _x + other._x ← allowed only inside the class

2) using operator+

3) two versions of * ← can't use isInstance since return types were diffence

Additional functions

(Not in the class)

```
// 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;
}
```

3.0 * pt;

Why?

overloading print

cout << pt;