CSE 333 – SECTION 7

Constructor insanity, C++ practice

Constructors

A constructor initializes a newly instantiated object

- a class can have multiple constructors
  - they differ in the arguments that they accept
  - which one is invoked depends on how the object is instantiated

You can write constructors for your object

- but if you don't write any, C++ might automatically synthesize a default constructor for you
  - the default constructor is one that takes no arguments and that calls default constructors on all non-POD* member variables (*POD = "Plain Old Data")
  - C++ does this **iff** your class has no const or reference data members, and no other user-defined constructors

Constructors

- You might choose to define multiple constructors:

```
Point::Point() {
  x_ = 0;
  y_ = 0;
}
Point::Point(const int x, const int y) {
  x_ = x;
  y_ = y;
}
void foo() {
  Point x;  // invokes the default (argument-less) constructor
  Point y(1,2);  // invokes the two-int-arguments constructor
}
```

Constructors

- You might choose to define only one constructor:

```
Point::Point(const int x, const int y) {
  x_ = x;
  y_ = y;
}
void foo() {
  // Compiler error; if you define any constructors, C++ will
  // not automatically synthesize a default constructor for
you.
  Point x;

  // Works.
  Point y(1,2);  // invokes the two-int-arguments constructor
}
```

Copy constructors

- **create a new object** as a copy of an existing object

```
Point::Point(const int x, const int y) : x_(x), y_(y) { }
Point::Point(const Point &copyme) {  // copy constructor
  x_ = copyme.x_;
  y_ = copyme.y_;
}
void foo() {
  // invokes the two-int-arguments constructor
  Point x(1,2);
  // invokes the copy constructor to construct y as a copy of x
  Point y(x);  // could also write as "Point y = x;"
}
```

When do copies happen?

The copy constructor is invoked if:

- you pass an object as a parameter
- to a call-by-value function

```
void foo(Point x) { ... }
Point y;  // default cons.
foo(y);   // copy cons.
```

- you return an object from a
- function

```
Point foo() {
  Point y;  // default cos.
  return y; // copy cons.
}
```

- you initialize an object from
- another object of the same type

```
Point x;   // default cons.
Point y(x);  // copy cons.
Point z = y; // copy cons.
```

But...the compiler is smart...

It sometimes uses a “return by value optimization” to eliminate unnecessary copies

- sometimes you might not see a constructor get invoked when you expect it

```
Point foo() {
  Point y;  // default constructor.
  return y; // copy constructor?  optimized?
}
Point x(1,2);     // two-ints-argument constructor.
Point y = x;      // copy constructor.
Point z = foo();  // copy constructor? optimized?
```

Synthesized copy constructor

If you don't define your own copy constructor, C++ will synthesize one for you

- it will do a shallow copy of all of the fields (i.e., member variables) of your class
- sometimes the right thing, sometimes the wrong thing

see SimplePoint.cc, SimplePoint.h

assignment != construction

- The “=” operator is the assignment operator
- - assigns values to an existing, already constructed object
- - you can overload the “=” operator

```
Point w;            // default constructor.
Point x(1,2);       // two-ints-argument constructor.
Point y = w;        // copy constructor.
y = x;              // assignment operator.
```

Operator Overloading

- A form of polymorphism.
- Give special meanings to operators in user-defined classes
- Special member functions in classes with a particular naming convention
- For E.g., for overloading the ‘=’ operator, define a member function named operator=

Overloading Operator =

```
Point &Point::operator=(const Point &pt) {
  if (this != &pt) { // if (this != &pt) { ... }" guard
    x_ = pt.x_;
    y_ = pt.y_;
  }
  return *this;
}
```

Rule of Three

If you define any of:

1. Destructor
2. Copy Constructor
3. Assignment (operator=)

Then you should normally define all three

Dealing with the insanity

C++ style guide tip

- if possible, disable the copy constr. and assignment operator

```
class Point {
 public:
  Point(int x, int y) : x_(x), y_(y) { }
 private:
  // disable copy cons. and "=" by declaring but not defining
  Point(Point &copyme);
  Point &operator=(Point &rhs);
};
Point w;        // compiler error
Point x(1,2);   // OK
Point y = x;    // compiler error
x = w;          // compiler error
```

Dealing with the insanity

C++11 adds new syntax

```
class Point {
 public:
  Point(int x, int y) : x_(x), y_(y) { }
  // declare copy cons. and "=" as deleted (C++11)
  Point(Point &copyme) = delete;
  Point &operator=(Point &rhs) = delete;
};
Point w;        // compiler error
Point x(1,2);   // OK
Point y = x;    // compiler error
x = w;          // compiler error
```

Dealing with the insanity

if you disable them, then you should instead probably have an explicit “CopyFrom” function

```
class Point {
 public:
  Point::Point(int x, int y) : x_(x), y_(y) { }
  void CopyFrom(const Point &copy_from_me);
 private:
  // disable copy cons. and "=" by declaring but not defining
  Point(const Point &copyme);
  Point &operator=(const Point &rhs);
};
```

```
Point x(1,2);          // OK
Point y(3,4);          // OK
x.CopyFrom(y);         // OK
```

Sec7 exercise

- Write a C++ program that:
- has a class representing a rectangle
  - Use SimplePoint/Point class to store the coordinates of the top-left corner
  - Has width, height
- has the following methods:
  - return the area of a rectangle
  - test if a point is inside a rectangle