CSE333 SECTION 7

Recall: Constructors

Initializes a newly instantiated object

```
// Default (zero-arg) Constructor
myClass::myClass() {
  <code>
}

myClass::myClass(arg, arg, …, arg) {
  <code>
}

// Copy Constructor
myClass::myClass(myClass other) {
  <code>
}
```

Aside: Where is this stuff?

Assume we have a class Point with a zero-argument constructor.

```
int main(int argc, char **argv) {
  Point p();
  Point *p2 = new Point;

  return 0;
}
```

What is p? What is p2?

Recall: Copy Constructor

Create a new object as a copy of an existing one

When do copies happen?

Copy Construction Usage

Parameter passing to call-by-value functions

Returning an object from a function

When you call the copy constructor

Is this always true?

Compiler Optimizations

Copy elision

- Compiler might optimize away unnecessary copying
- Even if the code executed by the Copy-Constructor changes the state of the program

Copy-Elision Example

```
#include <iostream>

using namespace std;

static int x = 0;

class Point {
 public:
  Point() : x_(0), y_(0) { }
  Point(Point &p) { x++; }
 private:
  int x_, y_;
};
```

```
int main(int argc, char **argv) {
  Point p1;  // Default constructor
  Point p2 = p1;  // Copy Constructor
  cout << x << endl;
  return 0;
}
```

What value(s) might this print?
Note: Most recent C++ 11 revision addresses this

Copy-Elision Example Continued

Possible values:

0. Why? If Copy-Elision takes place and the compiler decides that a copy is unnecessary it simply might not execute the code.

1. Why? If Copy-Elision doesn't take place the compiler will execute the copy constructor and increment x.

C++ Class Mystery

See problem 1 on worksheet

C++ Assignment

Assignment is done with the “=“ operator

Assigns values to an existing, already constructed object

C++ Assignment/Copy Constructor Example

See problem 2 on worksheet

Let's try and make a toString() equivalent

Java convention:

toString() – Returns a text representation of the class

We should have a text representation of our C++ classes.

How should we do it?

C++ iostream

We've been using it so far to print text to the console. We can also use it to print a text representation of our class.

But! iostream won't know what to print out for our class.

And! To encapsulate our data we should keep fields private

So, what do we do?

Friends

Are functions or classes declared with the friend keyword.

Friend allows non-member functions to access the private and protected members of a class.

Example:

```
friend std::ostream &operator<<(std::ostream &out,
                                const myClass &c);
friend std::istream &operator>>(std::istream &in,
                                const myClass &c);
```

Aside: Why do we want to use references here?

L-Values and R-Values

All values are either L-Values or R-Values

L-Values (locator values) – An object that occupies accessible memory

R-Values – All non-L-Values.

Example

```
int bar() { return 333; }

int main(int argc, char **argv) {
  bar() = 2;

  return 0;
}
```

Example

```
int course_num = 333;

int & course() { return course_num; }

int main(int argc, char **argv) {
  course() = 334;
  return 0;
}
```

Move constructor

Moves values from one object to another with copying (“steal” the resources)

Why would we use it? Optimize away temporary copies

Example: (Already #include and using …)

```
string retFoo(void) { string x(“foo”); return x; }
int main(int argc, char **argv) {
  string  b = move(retFoo());
  return 0;
}
```

Another example

```
#include <memory>    // for std::unique_ptr
#include <iostream>  // for std::cout, std::endl
#include <stdlib.h>  // for EXIT_SUCCESS

using namespace std;

int main(int argc, char **argv) {
  unique_ptr<int> x(new int(5));
  cout << "x: " << x.get() << endl;

  unique_ptr<int> y = std::move(x);  // y takes ownership, x abdicates it
  cout << "x: " << x.get() << endl;
  cout << "y: " << y.get() << endl;

  unique_ptr<int> z(new int(10));

  // z delete's its old pointer and takes ownership of y's pointer.
  // y abdicates its ownership.
  z = std::move(y);

  return EXIT_SUCCESS;
}
```

Assume x's original address is 0x1000000

Move Constructor

What do you think happens to the other value?

Move Constructor

What do you think happens to the other value?

Like (most) other things in C/C++. It depends!

Often it will leave the object getting “stolen” from in some valid but indeterminate state. Do not depend on this!

Move Constructor Syntax

Declaration:

className(className&& o);

Used when: An object is initialized from a temporary value of the same type:

Initialization: T a = std::move(b);

Argument passing: function(std::move(a));

Function return: return a, where the function declaration is T function(…) and T is the type of the move constructor

Let’s write a move constructor

See problem 4 on the worksheet

The Rule of 5

If you define any of:

- Destructor
- Copy Constructor
- Move Constructor
- Copy Assignment Operator
- Move Assignment Operator

You should write all 5.

The idea is that if you've written one of the functions above for your class, then the default versions of the other 4 are not sufficient.

Style tip

If possible, disable the copy constructor and the assignment operator.

We can do this by declaring the functions, but provide no definition of them.

If you do disable it, you should write a explicit CopyFrom function (same thing, different name)

This is not possible if you want to use your class in a STL container (it uses the copy constructor).