Review Sheet 4a: REVISED

CS 70: Data Structures and Program Development

Tuesday, February 11, 2019

Learning Targets

1. I can identify when objects are initialized and destroyed
2. I know the purpose of constructors, default constructors, destructors, copy constructors, and assignment operators
3. I can identify when these functions are implicitly called in a piece of code
4. I can use the default and delete keywords in a class declaration

A Simple Chicken Class

```
class Chicken {
    public:
        bool isHatched();

    private:
        bool hatched_;
};

int main() {
    Chicken henny;
    cout << henny.isHatched() << endl;
}
```

What happens?

Synthesized Default Constructor

```
Chicken::Chicken() {
    //All members are default-initialized.
    //Primitive type members have undefined value.
    //Object members are default-constructed.
    //Nothing more to do!
}
```

Our Own Default Constructor

```
class Chicken {
    public:
        Chicken();
...

Chicken::Chicken() : hatched_{false}
{}
```

Sometimes We Don't Want a Default Constructor

```
class Cow{
    public:
        Cow() = delete; //Don't synthesize
...

Cow bessy; //Will not compile!
```

A Problematic Barn : What's wrong?

Fixing the Barn

```
class Barn{
    public:
        Barn() = delete;
        Barn(size_t numCows, string filename);
    private:
        size_t numCows_;
        Cow** cowArr_;
};
```

What's the problem?

Synthesized Destructor

When an object is destroyed, its destructor is invoked.

```
Barn::~Barn() {
    //No special instructions
    //When this function returns
    //all data members are destroyed
    //(last to first)
}
```

Our Own Destructor

```
class Barn{
    public:
        ~Barn();
...

Barn::~Barn() {
    //Whatever needs to happen to clean up a barn
}
```

Sometimes We Want The Synthesized Destructor

```
class Cow{
    public:
        ~Cow() = default; //Synthesize
...
```

Exercise What functions are called on each line?

```
void barnyard() {
    Chicken* a = new Chicken{};
    Cow b{2, 3};
    Chicken c[3];
    Barn d{3, "cows.txt"};
    delete a;
}
```

Secret Cow Cloning Program

```
void printCow(Cow c) {
    cout << c.getNumSpots() << " " << c.getAge() << endl;
}
```

How does `c` get initialized?

Synthesized Copy Constructor

```
Cow::Cow(const Cow& other) :
    spots_{other.spots_}, age_{other.age_}
{
    //All data members are copy-constructed.
    //Nothing more to do!
}
```

(BTW, why can we access `other.spots_`?)

A Problematic Barn: What's wrong?

Our Own Copy Constructor

```
Barn::Barn(const Barn& other) :
    numCows_{other.numCows_}, cowArr_{new Cow*[numCows_]}
{
    //Whatever needs to happen to make the new Cow
    //a copy of other
}
```

Exercise What functions are called on each line?

```
Cow* cowsAbound(Cow a, Cow& b, Barn c) {
    Cow d{b};
    Cow* e = new Cow{2, 3};
    return e;
}

int main() {
    Cow w{4, 9};
    Cow x{2, 12};
    Barn y{4, "cowstats.txt"};
    Cow* z = cowsAbound(w, x, y);
    delete z;

    return 0;
}
```

Assignment Operator

```
Cow bessie{5, 8};
Cow bartholomoo{3, 10};
bartholomoo = bessie;
```

equivalent to...

```
bartholomoo.operator=(bessie);
```

Technically, `operator=` returns a reference to the object that was just modified. That's so you can do things like `x = y = z` (but don't do that).

Synthesized Assignment Operator

```
Cow& Cow::operator=(const Cow& rhs) {
    //Overwrite each data member
    numCows_ = rhs.numCows_;
    cowArr_ = rhs.cowArr_;

    //Return the object we just modified
    return *this;
}
```

Note: `this` is an implicit parameter to every member function. It stores the address of the object that the function was called on. So `*this` is a name for the object itself!

A Problematic Barn: What's wrong?

Our Own Assignment Operator

- There are subtle issues in writing an assignment operator
- There is an idiom that just works (relies on working copy constructor and destructor)
- Don't worry about it for now (examples in HW assignments later)

Summary

- Constructors
  - Invoked when an object is initialized
  - Set up the object's members
  - Which constructor is invoked depends on parameters
- Default Constructor
  - Invoked for default initialization
  - Constructor with no parameters
- Destructor
  - Invoked when an object is destroyed
  - Cleans up the object's members
  - Name is ~ClassName()
- Copy Constructor
  - Invoked when a copy is made (e.g. parameter passing)
  - Takes a const reference of the same type
  - Makes a copy (used for parameter passing etc.)
- Assignment operator
  - Invoked when an object is assigned to an existing object
  - Defined by a member function named `operator=`
  - Takes a const reference to the right hand side of =
  - Returns a reference to the object that was modified

Rules

Always define, default or delete

- The default constructor
- The destructor
- The copy constructor
- The assignment operator

The Rule of 3

- If you need to define one of these...
  - Destructor
  - Copy constructor
  - Assignment operator
- ...then you probably need to define them all
- (Otherwise probably default them all)
- Caveat: In HW we will often violate the rule of 3

Tricky Synax: What's happening here?

```
Cow bessie{5, 8};
Cow bartholomoo = bessie;
```

Exercise

What functions are called on each line?

```
void cowParty() {
    Cow a{4, 9};
    Cow b{2, 12};
    Cow c{a};
    Cow d = b;
    Cow& e = d;
    b = a;
    e = b;
    Barn f{4, "cowstats.txt"};
    Barn g{3, "cowlist.txt"};
    g = f;
}
```

Warning: What's happening here?

```
Cow bessie = Cow{5, 8}; //Don't ever write this!
```