CS 368 Announcements
Wednesday, March 13, 2013

Program p2

- due Monday, March 18
- note: will add in p3 destructor, copy constructor, operator=

Last Time

- wrap up intro to defining classes from last lecture
  - `const` member functions
- multi-file compilation - makefiles
- constructor
- member initialization

Today

- finish Ch. 4 – the Big Three
- copy constructor
- copy assignment (operator=)
- destructor
- Polynomial example

Next Time

- Ch. 5
- operator overloading
- `operatorX` syntax
- `explicit`
- conventions for arithmetic operators

Constructors

A constructor is called:

- when an object is declared:

- when an object is dynamically allocated:

In Java:
data members are initialized automatically before a constructor executes

In C++:
before body of constructor is executed, each data member (field) that is an object is initialized by calling the field's no-arg constructor

Copy Constructor

Called automatically when an object is:

- passed by value to a function

- returned (by value) as a function result

- declared with an initialization from an existing object of the same class

Copy Constructor (continued)

In IntList.cpp

```
IntList::IntList(const IntList &L) :
    items(new int[L.arraySize]), numItems(L.numItems),
    arraySize(L.arraySize)
{
    for (int k = 0; k < numItems; k++)
        items[k] = L.items[k];
}
```

Operator=

By default, just a shallow copy is done

- if have pointer data members, you'll want to write your own to do a deep copy

In IntList.cpp

```
IntList & IntList::operator=(const IntList &L) {

   if (this == &L)
       return *this;


   else {
       delete [] items;
       items = new int[L.arraySize];
       arraySize = L.arraySize;

       for (numItems = 0; numItems < L.numItems; numItems++)
           items[numItems] = L.items[numItems];
   }

   return *this;
}
```

Destructor

Called when object is about to go away:

- object (value parameter or local variable) goes out of scope
- pointer to object is deleted

In IntList.cpp

```
IntList::~IntList() {



}
```

testIntList.cpp

```
#include <iostream>
#include "IntList.h"

using namespace std;

void copyAgain(IntList L) {
    IntList Lnew = L;
    Lnew.print();
    L.print();
}



int main() {
    IntList L1;
    IntList L2(25);

    IntList *p, *q;
    p = new IntList;
    q = new IntList(25);

    for (int i = 0; i < 20; i++)
        L1.addToEnd(i+1);
    L1.print();

    L2 = L1;
    L2.addToEnd(-1);

    IntList L3(L1);

    L1.print();
    L2.print();
    L3.print();

    copyAgain(L3);

//  delete p;
//  delete q;

    return 0;
}
```

Polynomial.h

```
#ifndef POLYNOMIAL_H
#define POLYNOMIAL_H

#include
using namespace std;

class Polynomial {

    friend bool operator==(const Polynomial & lhs,
                           const Polynomial & rhs);

    public:

        // constructors
        Polynomial();
        Polynomial(double coefficients[], int number);
        Polynomial(const Polynomial & rhs);
        explicit Polynomial(double const_term);

        // destructor
        ~Polynomial();

        // named member functions
        int degree() const { return size - 1; }
        void print(ostream & out = cout) const;

        // assignment operators
        const Polynomial & operator= (const Polynomial & rhs);
        const Polynomial & operator+= (const Polynomial & rhs);
        const Polynomial & operator*= (double rhs);

    private:
        int size;   // size of the coefs array ( = degree + 1)
        double * coefs; // coefs will be an array
};

Polynomial operator+(const Polynomial & lhs,
                     const Polynomial & rhs);

Polynomial operator*(const Polynomial & lhs, double rhs);
Polynomial operator*(double lhs, const Polynomial & rhs);

ostream & operator<<(ostream & out, const Polynomial & p);

#endif
```

Constructors (in Polynomial.cpp)

```
// Constructor
// Creates a default polynomial p of the form p(x) = 0.0
Polynomial::Polynomial() : size(1), coefs(new double[1]) {
    coefs[0] = 0.0;
}



// Constructor
// Given array of coefs C (& it's size N) creates a polynomial
// p(x) = C[N-1]x^(N-1) + ... + C[2]x^2 + C[1]x + C[0]
Polynomial::Polynomial(double coefficients[], int number) :
  size(number), coefs(new double[number]) {
    for (int i = 0; i < size; i++) {
        coefs[i] = coefficients[i];
    }
}



// Constructor
// Given a constant term A, creates the polynomial p(x) = A
Polynomial::Polynomial(double const_term) :
  size(1), coefs(new double[1]) {
    coefs[0] = const_term;
}
```

The "Big Three" (in Polynomial.cpp)

```
// Copy constructor
// Creates a polynomial from the given polynomial
Polynomial::Polynomial(const Polynomial & rhs) :
  size(rhs.size), coefs(new double[rhs.size]) {
    for (int i = 0; i < size; i++) {
        coefs[i] = rhs.coefs[i];
    }
}



// Destructor
Polynomial::~Polynomial() {
    delete [] coefs;
}



// Overload assignment =
const Polynomial & Polynomial::operator=(
                                        const Polynomial & rhs) {
    if (this == &rhs) {
        return *this;
    }

    else {
        delete [] coefs;
        coefs = new double[rhs.size];
        size = rhs.size;
        for (int i = 0; i < size; i++) {
            coefs[i] = rhs.coefs[i];
        }
    }
    return *this;
}
```