

# CS 368 Announcements

## Wednesday, October 23, 2013

**Record your attendance on the sign-in sheet.**

### **Program p2**

- due TODAY, October 23rd by 10:00 pm

### **Program p3**

- assigned
- add destructor, copy constructor, operator= to p2 code

### **Last Time**

- wrap up intro to defining classes
- compiling multiple files

### **makefiles Today**

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

### **Next Time**

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

## Recall IntList.h

```
#ifndef INTLIST_H
#define INTLIST_H

class IntList {

public:
    // Constructors
    IntList();
    IntList(int size);

    IntList(const IntList &L);           // copy constructor
    ~IntList();                          // destructor
    IntList & operator=(const IntList &L); // assignment

    // Other public member functions
    void addToEnd(int k); // adds value k to end of list
    void print() const;  // prints out list (using cout)

private:
    static const int SIZE = 10; // default init array size
    int *items;                // dynamically allocated array of ints
    int numItems;              // number of items currently in list
    int arraySize;            // current size of the array
};

#endif
```



## 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;
}
```

## Member Initialization Lists

### In IntList.cpp

```
IntList::IntList() :  
    items(new int[SIZE]), numItems(0), arraySize(SIZE)  
{  
}
```

```
IntList::IntList(int size) :  
    items(new int[size]), numItems(0), arraySize(size)  
{  
}
```

## Copy Constructor

**Automatically called 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)

- default version that's automatically provided only does a shallow copy
- if your class has pointer data members using heap allocation, you'll want to write your own to do a deep copy

### 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=

**Called when the assignment operator is used with objects.**

- default version that's automatically provided only does a shallow copy
- if your class has pointer data members using heap allocation, 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

**Automatically called when an object is about to “go away”:**

- object (value parameter or local variable) goes out of scope
- pointer to object is deleted
  
- if your class has pointer data members using heap allocation, you'll want to delete their heap memory

**In IntList.cpp**

```
IntList::~IntList() {  
  
  
}
```

## 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
//  $p(x) = C[N-1]x^{N-1} + \dots + C[2]x^2 + C[1]x + C[0]$ 
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;
}
```