

# CS 368 Announcements

## Wednesday, October 30, 2013

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

### **Program p3**

- due Wednesday, November 6th
- add destructor, copy constructor, operator= to classes from p2

### **Last Time**

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

### **Today**

- Unix utilities: gdb and valgrind
- start Ch. 5
- `const` member functions
- operator overloading
- `operatorX` syntax
- overloading assignment ops
- overloading arithmetic ops

### **Next Time**

- continue Ch. 5
- `explicit`
- overloading output operator (<<)
- overloading relational ops
- `friend`
- overloading increment (++) and decrement (--)

## Unix Utilities: gdb and valgrind

**`gdb`**

**`valgrind`**

### **`valgrind` and Makefiles**

```
main: testIntList.o IntList.o
    g++ testIntList.o IntList.o

testIntList.o: testIntList.cpp IntList.h
    g++ -c testIntList.cpp

IntList.o: IntList.cpp IntList.h
    g++ -c IntList.cpp

valgrind: main
    valgrind --leak-check=full main

clean:
    rm *.o
```

## Recall 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
    Polynomial & operator= (const Polynomial & rhs);
    Polynomial & operator+= (const Polynomial & rhs);
    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
```

# Operator Overloading

Why overload operators?

Operators that cannot be overloaded:

Operators that shouldn't be overloaded:

(Non-obvious) operators that can be overloaded:

## Can't

- create new operators
- change precedence or associativity
- change arity (# of parameters)

## Keep in mind

- don't overload just because you can
- be aware of commutative (symmetric) operations
- be aware of what happens if operands are of different types

## Polynomial.h (Operator Overloading)

```
#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
    Polynomial & operator= (const Polynomial & rhs);
    Polynomial & operator+= (const Polynomial & rhs);
    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
```

## Options for Operator Overloading

**Member Function:**

**Non-member Function:**

**`friend` Function:**

## Assignment Ops: 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
    Polynomial & operator= (const Polynomial & rhs);
    Polynomial & operator+= (const Polynomial & rhs);
    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
```

## Assignment Ops: Member vs. Non-member Function?

**Consider:**

```
Polynomial p1, p2;
```

```
p1 = 1.1;
```

```
p2 += p1;
```



## Assignment Ops: Polynomial.cpp

```
// Overload assignment +=
Polynomial & Polynomial::operator+= (
                                const Polynomial & rhs) {
    int newSize = (rhs.size > size) ? rhs.size : size;
    double *newCoef = new double [newSize];

    for (int i = 0; i < newSize; i++) {
        newCoef[i] = 0;
    }
    for (int i = 0; i < rhs.size; i++) {
        newCoef[i] += rhs.coefs[i];
    }
    for (int i = 0; i < size; i++) {
        newCoef[i] += coefs[i];
    }
    delete [] coefs;
    coefs = newCoef;
    return *this;
}

// Overload assignment *= so it supports scalar multiplication
Polynomial & Polynomial::operator*=(double rhs) {
    for (int i = 0; i < size; i++) {
        coefs[i] *= rhs;
    }
    return *this;
}
```

## Arithmetic Ops: 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
    Polynomial & operator= (const Polynomial & rhs);
    Polynomial & operator+= (const Polynomial & rhs);
    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
```

## Arithmetic Ops: Member vs Non-member Function?

**Consider:**

```
Polynomial p1 = 11.22, p2;
```

```
p2 = p1 + 1.1;
```

```
p2 = 1.1 + p1;
```

## Arithmetic Ops: Polynomial.cpp

```
// Overload +
Polynomial operator+(const Polynomial & lhs,
                    const Polynomial & rhs) {
    Polynomial answer(lhs);
    answer += rhs;
    return answer;
}

// Overload * so it supports scalar multiplication. Note that
// we overload it twice so we can do either:
//     polynomial * scalar
// or
//     scalar * polynomial

Polynomial operator*(const Polynomial & lhs, double rhs) {
    Polynomial answer(lhs);
    answer *= rhs;
    return answer;
}

Polynomial operator*(double lhs, const Polynomial & rhs) {
    Polynomial answer(rhs);
    answer *= lhs;
    return answer;
}
```