CS368 Lecture 9

Wednesday 29 October 2014

Reminders:

- P3 due Wednesday 5 Nov
- Reading: Chap. 4 complete, begin Chap. 5

Last class:

- The “Big 3”
    - Copy Constructor
    - Destructor
    - Assignment operator overloading

Today:

- More overloading operators
- Makefiles

Recall the Big 3

Destructor

Copy Constructor

'=' operator

More operatorX overloading: The Polynomial Class

Adding polynomials:

```
p3 = p1 + p2;
p4 += p5;
```

Multiplying polynomials:

```
p2 = p1 * 5.0;
p2 = 5.0 * p2;

p3 *= 5.0
```

“Incrementing” polynomials:

```
p1++;
++p1;
```

Overloading `operator*=`

Use:

```
Polynomial p1([3.0, 1.0], 2);
p1 *= 10;
```

OR

```
p1.operator*=(10.0);
```

In the header file:

```
class Polynomial {

    …

    Polynomial& operator*=(double rhs);

}
```

In the source file:

```
Polynomial& Polynomial::operator*= (double rhs) {

    for (int i=0; i < size; i++)
        coefs[i] *= rhs;

    return *this;
}
```

Overloading `operator+=`

Use:

```
Polynomial p1([3.0, 1.0], 2);
Polynomial p2([2.0], 1);

p1 += p2;
```

OR

```
p1.operator+=(p2);
```

In the header file:

```
class Polynomial {

    …

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

}
```

In the source file:

```
Polynomial& Polynomial::operator+=(const Polynomial& rhs) {

    int resultsize = (rhs.size > size) ? rhs.size : size;
    double* resultcoefs = new double[resultsize];

    for (int i=0; i < rhs.size; i++)
        resultcoefs[i] += rhs.coefs[i];

    for (int i=0; i < size; i++)
        resultcoefs[i] += coefs[i];

    delete[] coefs;
    coefs = resultcoefs;
    size = resultsize;
    return *this;
}
```

Overloading `operator+` and `operator*`

Re-use code

Non-member functions

Return value

Use:

```
p1 = 3.4 * p2;
p1 = p2 * 3.4
```

In header file:

```
// outside class declaration!

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

In source file:

```
// overload polynomial * double
Polynomial operator* (const Polynomial& lhs, double rhs){
    Polynomial answer(lhs);    // calls copy constructor
    answer *= rhs;             // calls operator *=
    return answer;
}

// overload double * polynomial
Polynomial operator* (double lhs, const Polynomial& rhs){
    Polynomial answer(rhs);    // calls copy constructor
    answer *= lhs;             // calls operator *=
    return answer;
}
```

`operator+` is similar:

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

Overloading `operator++`

Use:

```
(p1++).print(); // postfix
(++p2).print(); // prefix
```

In header file:

```
// Unary operators
const Polynomial& operator++(); // prefix
Polynomial operator++(int); // postfix
```

In source file:

```
// prefix overloading: make change, then return
const Polynomial& Polynomial::operator++() {
    coefs[0] = coefs[0] + 1.0;
    return *this;
}
```

Fake `int` as parameter:

```
// postfix overloading: save old, change, return old
Polynomial Polynomial::operator++(int) {
    Polynomial temp = *this;      // save old
    coefs[0] = coefs[0] + 1.0;    // change
    return temp;                  // return old
}
```

Makefiles