CSCI 2270
Data Structures and Algorithms
BigNum answers
finish polynomial part 1

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 112/128
Wed 9:30am-11:30am
Thurs 10am-11am

Admin

Exam 2 is graded… gently

BigNum part 2 solutions are posted.

Grading interviews for part 2 are up; I have enough slots for everyone, but it’s first come, first served. We won’t be setting aside more interviews if you miss one…

BigNum deep copies

Copy constructor and assignment operator

Similar code (both make a deep copy)

Operator = has to handle 2 extra things:

Check for self assignment

Delete preexisting digits array

Copy constructor doesn’t need to worry about these extra cases

--why not?

BigNum solution,operator =

Copy constructor and assignment operator (similar…)

```
BigNum& BigNum::operator=(const BigNum& anotherBigNum)
{
        if (this == &anotherBigNum)  return *this;
        if (digits != nullptr) delete [] digits;
        capacity = anotherBigNum.capacity;
        digits = new unsigned int[capacity];
        positive = anotherBigNum.positive;
        used = anotherBigNum.used;
        copy(anotherBigNum.digits, anotherBigNum.digits +
used, digits);
        return *this;
}
```

BigNum solution copy constructor

Copy constructor and assignment operator (similar…)

```
BigNum::BigNum(const BigNum& anotherBigNum)
{
        digits = nullptr;
        // makes operator = do the work; use that if you use this
        *this = anotherBigNum;
}
```

What’s this line

```
        digits = nullptr;
```

make the operator = do when it runs?

BigNum return types

Operator +=, returns changed BigNum by reference

```
    BigNum& BigNum::operator+=(const BigNum& addend)
    {
        …
        return *this;
    }
```

Operator +, returns a BigNum by value

```
    BigNum operator+(const BigNum& a, const BigNum& b)
    {
        BigNum result;
        …
        return result;
    }
```

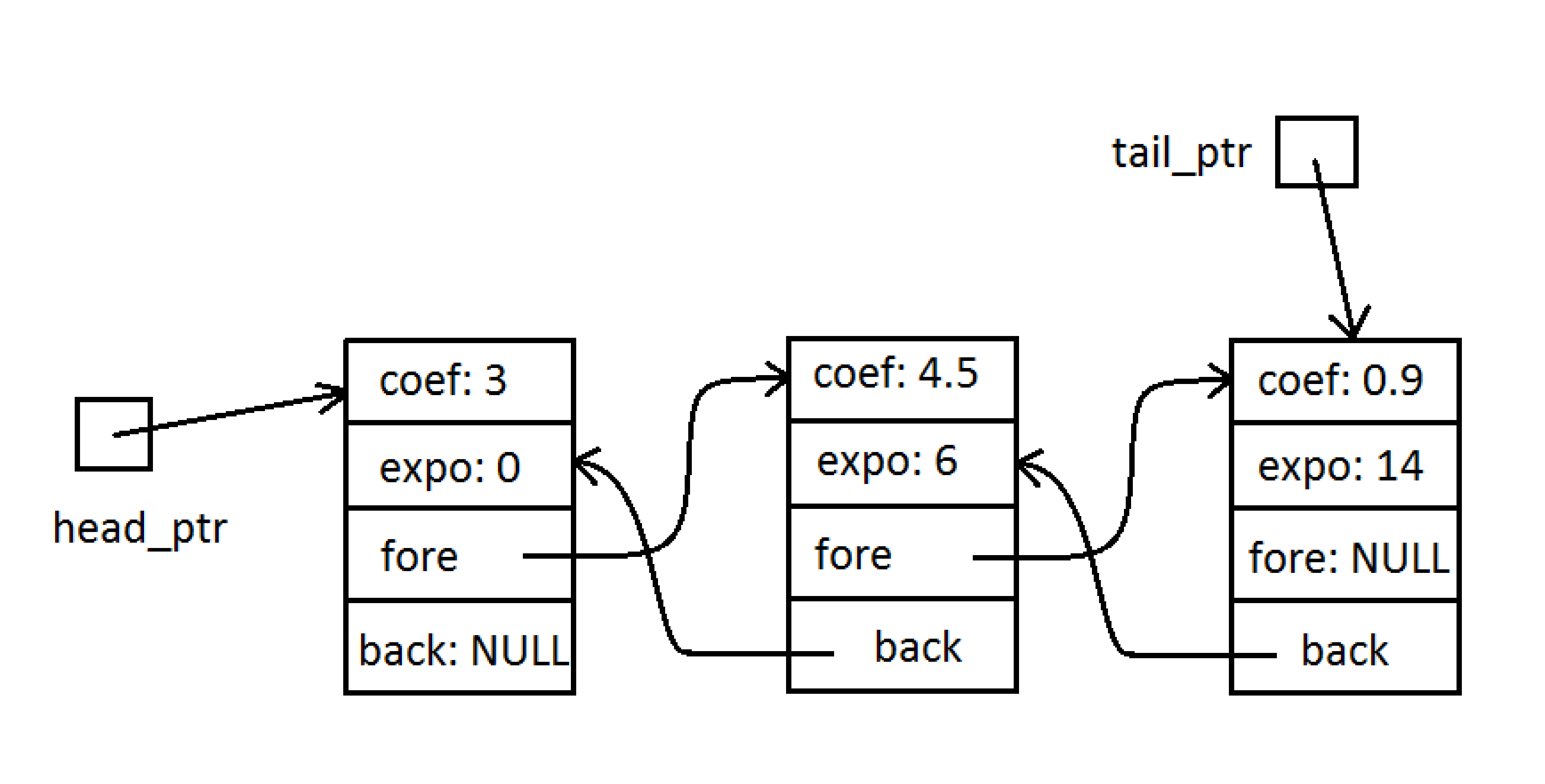
A new and interesting bug…

Dangling reference…

What if operator += returns a local BigNum by reference?

```
BigNum& BigNum::operator+=(const BigNum& addend)
{
        BigNum result;
        …
        return result;            <- BAD IDEA
}
```

Polynomial as DLL



p = 3x^0 + 4.5x^6 + 0.9x^14

Set_recent(exponent)

Move the recent_ptr to either

the node with the matching exponent, if it’s in the list, or
the node with the next highest exponent

Suppose the polynomial is 0x^0 + 6x^7 + 19x^10

4 cases; get these working early on:

exponent == 0

recent_ptr = head_ptr;

exponent == current_degree (here, this is 10)

recent_ptr = tail_ptr;

exponent > 0 && exponent < current_degree && exponent’s in list (set_recent(7); chooses the 6x^7 node)

exponent > 0 && exponent < current_degree && exponent’s not in list (set_recent(8); chooses the 6x^7 node too)

Coefficient(exponent)

Use set_recent to get the coefficient of a node with an exponent

```
set_recent(exponent);
```

If a node with that exponent exists, return its coefficient

Else return 0.0

next_term and previous_term

next_term(exponent)

Gets the next term in the list with a higher exponent and a non-zero coefficient, and returns that higher exponent.

If there is no such term, it returns 0

previous_term(exponent)

Gets the previous item in the list with a lower exponent and a non-zero coefficient, and returns that lower exponent.

If there is no such term, it returns UINT_MAX (rollover!)

Most of your other functions (operator <<, operator +, etc.) use these two helper functions

Output operator

Do-while loop is useful here

executes once, then checks whether to continue

```
ostream& operator << (ostream& out, const polynomial& p)
{
        unsigned int expo = 0;
        do
        {
                out << p.coefficient(expo) << "*x^" << expo;
                expo = p.next_term(expo);
                if (expo != 0) out << " + ";
        } while(expo != 0);
}
```

Add_to_coef; tricky cases

We might add to the coefficient for the x^0 node

That x^0 node's always there; just change its coefficient

Suppose our polynomial has a node for 5x^7

If we add -5 to the x^7 coefficient, then we have to remove this node (5 + -5 = 0)

If we add 2 to the x^7 coefficient, we just change the coefficient to become 7 (5 + 2 = 7)

Suppose our polynomial never had an x^7 node before

If we add 5 to this coefficient, then we'll add a new node for this term (0 + 5 = 5)

Assign_coef; tricky cases

We might assign a coefficient to the x^0 node

That x^0 node's always there; just change its coefficient

We might assign a coefficient of 0 to the x^7 node

If we had an x^7 node with a non-zero coefficient,

now we have to remove this node

If we never had an x^7 node, nothing will change

What if we assign a coefficient of 6 to the x^7 node, and the x^7 node wasn't there before?

Now we add a node for this coefficient of x^7