자료구조 HW2

B935394 컴퓨터공학과 장준희

September 28, 2020

1 개념 설명

1.1 연산자 오버로딩

연산자 오버로딩은 피연산자에 따라 다른 연산을 하도록 동일한 연산자를 중복해서 작성하는 것이다.
returnType operator operatorSymbol(list of variable) 의 형태로 선언한다.

2 코드 설명

2.1 <<연산자

```
ostream& operator<< (ostream& os, Polynomial& p) {
        for (int i = 0; i < p.terms; ++i) {
                if ((p.termArray[i].coef != 0) && (p.termArray[i].coef != 1)) {
                        if (p.termArray[i].coef == -1)
                                os << "-";
                        else
                                os << p.termArray[i].coef;
                }
                if (p.termArray[i].exp != 0) {
                        os << "x";
                        if (p.termArray[i].exp != 1)
                                os << "^" << p.termArray[i].exp;
                }
                if (i + 1 < p.terms) {
                        if (p.termArray[i + 1].coef >= 0)
                                os << "+";
                        else if (p.termArray[i + 1].coef)
                                os << "";
                }
        }
        os << endl;
        return os;
}
```

일단 p의 모든 원소를 읽어야하기 때문에 for문으로 다 읽기 시작했다.
첫번째 if :: 계수가 0,1이 아니면 계수를 출력해준다. 다만 계수가 -1이면 -만을 출력해준다.
두번째 if :: 지수가 0이 아니면 일단 x를 출력하고, 지수가 1이면 그대로, 지수가 1이 아니면 삿갓표와 지수를 출력한다
세번째 if :: 배열의 다음 원소가 존재하면, 그 원소의 계수에 따라 적절한 +연산자 혹은 공백을 출력한다.

2.2 NewTerm함수

```
void Polynomial::NewTerm(const float theCoeff, const int theExp) {
        if (this->terms == this->capacity) {
```

```
                this->capacity += 1;
                Term*tempTerm = new Term[capacity];
                for (int i = 0; i < terms; ++i) {
                        tempTerm[i] = this->termArray[i];
                }
                delete[] termArray;
                this->termArray = tempTerm;
        }
        this->termArray[this->terms].coef = theCoeff;
        this->termArray[this->terms++].exp = theExp;
}
```

만약 다항식 배열에 주어진 항이 없다면, 새로 추가하는 함수이다. 만약 배열이 꽉차있다면, 임시로 배열을 선언하고 복사를 한 후, 임시배열을 반환한다. 그리고 새로 생긴 항 또는 있던 항에 계수와 지수를 넣어준다.

2.3 +연산자

```
Polynomial Polynomial::operator+(Polynomial& b) {
        Polynomial tempPoly; int i = 0; int j = 0; float c = 0;
        while ((i <this->terms) && (j < b.terms)){
                if (this->termArray[i].exp == b.termArray[j].exp) {
                        c = this->termArray[i].coef + b.termArray[j].coef;
                        if (c != 0) tempPoly.NewTerm(c, this->termArray[i].exp);
                        i++; j++;
                }
                else if (this->termArray[i].exp > b.termArray[j].exp) {
                        tempPoly.NewTerm(this->termArray[i].coef, this->termArray[i].exp);
                        i++;
                }
                else if (this->termArray[i].exp < b.termArray[j].exp) {
                        tempPoly.NewTerm(b.termArray[j].coef, b.termArray[j].exp);
                        j++;
                }
        }
        for (; i < this->terms; ++i) {
                tempPoly.NewTerm(this->termArray[i].coef, this->termArray[i].exp);
        }
        for (; j < b.terms; ++j) {
                tempPoly.NewTerm(b.termArray[j].coef, b.termArray[j].exp);
        }
        return tempPoly;
}
```

반환할 임시다항식을 선언한다. while문에서는 두 다항식 중 하나의 다항식이 다 읽혀질 때까지, 두 지를 비교해, 같다면 두 계수를 더한 값과 그 지수를, 만약 한쪽이 더 크다면 NewTerm함수를 활용해 새로이 지수와 그 계수를 대입한다.

2.4 /연산자

```
Polynomial Polynomial::operator/(Polynomial& b)
{
        Polynomial c;
        bool exist=false;
        int maxExp = this->termArray[0].exp;
        float devider = -b.termArray[1].coef;
        float subSum;
        int i = 0; int j = 0; int h = 0;
        //나눗셈에 분배법칙을 적용하려 했으니 될리가 없었다.
        //다행히 분모가 1차로 제한돼있으므로 조립제법을 설계하자.
        for (; i < maxExp ; ++i) {
                if (i == 0) {
                        c.NewTerm(this->termArray[i].coef, maxExp - 1);
                        subSum = devider * this->termArray[i].coef;
                }
                else {
                        for (; j < this->terms; ++j) {
                                if (maxExp - i == this->termArray[j].exp) {
                                        exist = true;
                                        break;
                                }
                        }
                        if (exist == true) {
                                c.NewTerm(this->termArray[i].coef + subSum, maxExp - (i+1));
                                subSum = (this->termArray[i].coef + subSum)*devider;
                        }
                        else if (exist == false) {
                                c.NewTerm(subSum, maxExp - (i+1));
                                subSum = subSum * devider;
                        }
                }
        }
        return c;
}
```

멍청하게도... 나눗셈이 분배법칙이 가능하리라 믿어버렸다. 머리속에서 생각하면 항상 위험하다는 교훈을 얻었다. 나누는 다항식이 2차만 넘겼어도 복잡할 것 같았는데, 다행히 1차로 한정되어있어서 조립제법을 이용해 나눗셈을 하기로 생각했다. 1차로 정해져있기 때문에 devider를 선언할 때 인덱스로 값에 접근해 대입할 수 있었다. subSum은 조립제법에서 지그재그로 왔다갔다 하는 수를 표현해보았다. 조립제법 원리에 따라서 최고차항의 계수는 그냥 내려준다. 그 다음부터는 else블록 안의 내용을 실행한다. maxExp-i는 else 블록 안에서 그 밖의 for문이 돌 때마다 하나씩 줄어드는(조립제법은 건너뛴 지수는 0으로 표현해야하니, 하나씩 줄어들게 하였다.) 고정된 값이고, 이를 나눠지는 다항식의 모든 지수와 비교하여 만약 존재한다면 아래의 if 블록을 존재하지 않는다면 else if 블록을 실행하게 하였다. if문의 경우 지수가 존재하기 때문에, 그 지수 항의 계수와 subSum을 더해준다. 그리고 다음 subSum을 앞서 합한 값과 devider를 곱하여 대입한다.