# 자료구조 HW2

B711222 박조은
Hongik University
mrnglory@mail.hongik.ac.kr

October 13, 2019

## I. List of Source Files

- hw2a
  - polya.h
  - polya.cpp
  - hw2a.cpp
- hw2b
  - polyb.h
  - polyb.cpp
  - hw2b.cpp

## II. hw2a

### i. polya.h

```
1  #include <iostream>
2  #ifndef POLYNOMIAL_H
3  #define POLYNOMIAL_H
4  using namespace std;
5
6  class Polynomial;
7  class Term {
8  public:
9          friend class Polynomial;
10         friend ostream& operator << (ostream&,
               ↪ Polynomial&);
11         friend istream& operator >> (istream&,
               ↪ Polynomial&);
12
13 private:
14         float coef;
15         int exp;
16 };
17
18 class Polynomial {
19 public:
20         Polynomial();
21         Polynomial operator + (Polynomial&);
22         void NewTerm(const float, const int);
23 friend ostream& operator << (ostream&, Polynomial&);
24 friend istream& operator >> (istream&, Polynomial&);
25
26 private:
27         Term *termArray;
28         int capacity;
29         int terms;
30 };
31 #endif
```

```
/*
 * line 2, 3, 31: 헤더파일 중복으로 발생할 수 있는 문제를 막기 위해 #ifndef #endif #define 전처리기 사용
 * line 7: Term class 생성
 * line 6: 밑의 class Term 에서 Polynomial 선언 가능케 하도록 미리 호출
 * line 9 - 11: 다른 클래스의 friend로 선언된 멤버들간에 상호 접근할 수 있게 friend 키워드 선언
 * line 10: output operator overloading
 * line 11: input operator overloading
 * line 13 - 15: private 타입의 멤버변수 coef, exp 는 각각 polynomial 의 계수와 지수를 지칭
 * line 20: 다항식 p(x) = 0 을 생성
 * line 21: plus operator overloading
 * line 22: 새로운 항의 추가 및 배열의 크기를 두 배로 확장하는 기능을 가진 public 타입의 멤버 함수 선언
   (polya.cpp file 의 line 71 - 84 에서 구현)
 * line 26 - 29: Polynomial의 전용 데이터 멤버 선언
 * line 27: 0 이 아닌 항의 배열
 * line 28: termArray 의 크기
 * line 29: 0 이 아닌 항의 갯수
*/
```

## ii. polya.cpp

```
#include <iostream>
#include "polya.h"
using namespace std;

istream& operator >> (istream& is, Polynomial& p)
{
	int noofterms;
	float coef;
	int exp;

	is >> noofterms;

	for (int i = 0; i < noofterms; i++)
	{
		is >> coef >> exp;
		p.NewTerm(coef, exp);
	}
	return is;
}

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

		if (p.termArray[i].coef < 0)
		{
			if (p.termArray[i].coef != -1)
				os << p.termArray[i].coef;
			else
				os << "-";
		}

		if (p.termArray[i].exp != 0)
			os << "x^" << p.termArray[i].exp;

		for (int j = i + 1; j < p.terms; j++)
		{
			if (p.termArray[i].exp == p.termArray[j].exp)
			{
				p.termArray[i].coef += p.termArray[j].coef;
				p.termArray[j] = p.termArray[p.terms - 1];
				p.terms--;
			}
		}

	}

	os << endl;
	return os;
}

Polynomial::Polynomial():capacity(1), terms(0)
{
	termArray = new Term[capacity];
}

void Polynomial::NewTerm(const float theCoef, const int theExp) // theCoeff -> theCoef
{
	if (terms == capacity)
	{
		capacity *= 2;
		Term * temp = new Term [capacity];
		copy(termArray, termArray + terms, temp);
		delete [] termArray;
		termArray = temp;
	}

	termArray[terms].coef = theCoef;
	termArray[terms++].exp = theExp;
}

Polynomial Polynomial::operator + (Polynomial& b)
{
	Polynomial c;
	int aPos = 0, bPos = 0;

	while ((aPos < terms) && (bPos < b.terms))
	{
		if (termArray[aPos].exp == b.termArray[bPos].exp)
		{
```

```
95                          float t = termArray[aPos].coef +
                                ↪ b.termArray[bPos].coef;
96                          if (t)
97                                  c.NewTerm(t, termArray
                                        ↪ [aPos].exp);
98                          aPos++;
99                          bPos++;
100                 }
101                 else if (termArray[aPos].exp < b.
                          ↪ termArray[aPos].exp)
102                 {
103                         c.NewTerm(b.termArray[bPos].
                                ↪ coef, b.termArray[bPos
                                ↪ ].exp);
104                         bPos++;
105                 }
106                 else
107                 {
108                         c.NewTerm(termArray[aPos].
                                ↪ coef, termArray[aPos].
                                ↪ exp);
109                         aPos++;
110                 }
111         }
112
113         for (; aPos < terms; aPos++)
114                 c.NewTerm(termArray[aPos].coef,
                          ↪ termArray[aPos].exp);
115
116         for (; bPos < terms; bPos++)
117                 c.NewTerm(b.termArray[bPos].coef, b.
                          ↪ termArray[bPos].exp);
118
119         for (int i = 0; i < c.terms; i++) // 내림차순 정렬
120                 for (int j = i + 1; j < c.terms; j++)
121                         if (c.termArray[i].exp < c.
                                ↪ termArray[j].exp)
122                         {
123                                 Term temp = c.
                                        ↪ termArray[i];
124                                 c.termArray[i] = c.
                                        ↪ termArray[j];
125                                 c.termArray[j] = temp;
126                         }
127
128         return c;
129 }
```

/*

* line 5 - 19: poly.in file 에 적힌 값을 불러 들여서 각각 항의 갯수, 계수 및 지수라는 정보로 저장하는 과정
* line 7: 항의 갯수 정보 값 저장 공간
* line 8: 각 항의 계수 정보 값 저장 공간
* line 9: 각 항의 지수 정보 값 저장 공간
* line 13: 각 항의 계수 및 지수 정보를 들여오는 반복문의 횟수를 해당 다항식의 항의 갯수인 noofterms 의 값 만큼 실행
* line 21 - 64: 입력 받은 polynomial 을 실제 수식의 형태로 경우에 따라 적절히 출력 해주는 함수 구현
* line 23: polynomial 의 항의 갯수 만큼 반복문 수행
* line 25: 계수가 양수일 경우
* line 27: 초항이 아닐 경우
* line 29 - 30: 계수가 1 이 아닐 경우, + 기호와 계수를 함께 출력한다.
* line 31: 계수가 1 일 경우, 계수를 출력하지 않고 + 기호만 출력한다.
* line 34: 초항일 경우
* line 35 - 36: 계수가 1이 아닐 경우, + 기호 없이 계수만 출력한다.
* line 39: 계수가 음수일 경우
* line 41: 계수가 -1 이 아닐 경우
* line 42: 계수에 - 부호가 이미 붙어있으므로 계수 자체를 출력한다.
* line 43: 계수가 -1 일 경우
* line 44: - 부호만 붙인다.
* line 47: 지수가 0 이 아닐 경우, 즉 상수항이 아닐 경우
* line 48: x 의 exp 승의 표현 형태를 출력한다.
* line 50: 같은 차수의 항끼리 합쳐서 하나의 항으로 표현하는 코드
* line 52: 서로 비교한 두 개의 항이 같은 차수일 경우
* line 54: 두 항의 계수를 합하여 index i 에 저장, 고로 하나의 항으로 합치기 위한 과정
* line 55: 차수가 같은 두 항 중 index j 항은 이미 index i 항과 합쳤으므로, 더 이상 고려 대상이 아니다. 따라서 index j 항을 삭제 해야 하며, 다항식 맨 뒤에 존재하고 있는 항, 즉 index p.terms - 1 항을 index j 항에 복사하여 index j 항을 날린다.
* line 56: 그리고 이미 index j 항에 복사 한 index p.terms - 1 항을 제거한다.
* line 66: Polynomial 의 기정 생성자가 capacity 와 terms 를 각각 0 과 1 로 초기화한다.

* line 68: capacity 의 크기인 Term 타입의 배열 termArray
* line 71: 새로운 항을 termArray 끝에 추가한다.
* line 73: termArray 가 꽉 찰 경우
* line 75: 배열의 크기를 나타내는 값 capacity 를 두 배로 확장한다.
* line 76: 크기가 capacity 인 Term 타입의 배열 temp 를 임시로 생성한다.
* line 77: 배열 temp 에 termArray 의 모든 원소들을 복사한다.
* line 78: termArray 의 이전 메모리를 반환한다.
* line 79: 배열 temp 에 임시로 복사해둔 모든 항들을 다시 배열 termArray 에 저장한다.
* line 82: 배열 termArray 에 새로운 계수 정보를 추가한다.
* line 83: 배열 termArray 의 항의 갯수를 증가시킨 자리에 새로운 지수 정보를 추가한다.
* line 86: *this 와 b 를 더한 결과를 반환한다.
* line 88: Polynomial 타입의 다항식 c 를 생성한다.
* line 89: 다항식 *this 와 b 의 각 항들의 계수 및 지수를 비교하기 위해 index 값을 나타내는 a position, b position 변수를 설정 및 초기화한다.
* line 93: *this 와 b 가 같은 차수의 항이 있을 경우
* line 95: 각 계수의 값들을 더한 결과를 실수형 변수 t 에 저장한다.
* line 96: 계수가 t 인 해당 차수 항을 다항식 c 에 새로운 항으로 추가한다.
* line 98, 99: aPos, bPos 를 1씩 증가시킨다.
* line 101 - 109: 다항식 c 에 차수를 내림차순으로 차곡차곡 저장한다.
* line 113 - 114: *this의 나머지 항들을 다항식 c 에 추가한다.
* line 116 - 117: 다항식 b 의 나머지 항들을 다항식 c 에 추가한다.
* line 119 - 126: c 에 새로 추가된 항들을 다시 전체적으로 내림차순 정렬 시켜준다.

*/

## iii. hw2a.cpp

```
1  #include <iostream>
2  using namespace std;
3  #include "polya.h"
4  int main()
5  {
6          Polynomial p1, p2;
7
8          cin >> p1 >> p2;
9          Polynomial p3 = p1 + p2;
10         cout << p1 << p2 << p3;
11 }
```

/*

* line 6: Polynomial class 의 instance 로 p1, p2 를 정의한다.
* line 8: instance p1, p2 에 해당되는 다항식을 구성하는 값들을 입력 받아온다.
* line 9: 입력받은 두 개의 다항식을 덧셈하여 세로운 다항식에 저장한다.
* line 10: instance p1, p2 에 해당되는 다항식 및 다항식 덧셈 결과를 출력한다.

*/

## iv. Results

### iv.1 makefile

```
1  hw2a:hw2a.o polya.o
2          g++ -o hw2a hw2a.o polya.o
3  hw2a.o polya.o:polya.h
```

### iv.2 poly.in

```
1  3 3.0 5 2.0 3 -4.0 0
2  4 1.0 8 -7.0 5 -1.0 3 -3.0 0
```

### iv.3 compile

```
1  [B711222@localhost hw2d]$ ./hw2a < poly.in
2  3x^5+2x^3-4
3  x^8-7x^5-x^3-3
4  x^8-4x^5+x^3-7
```

# III. HW2B

## i. polyb.h

```
#ifndef POLYNOMIAL_H
#define POLYNOMIAL_H
#include <iostream>
using namespace std;

class Polynomial;
class Term {
friend class Polynomial;
friend ostream& operator << (ostream&, Polynomial&);
friend istream& operator >> (istream&, Polynomial&);

private:
        float coef;
        int exp;
};

class Polynomial {
public:
        Polynomial();
        Polynomial operator + (Polynomial&);
        Polynomial operator * (Polynomial&);
        void NewTerm(const float, const int);

friend ostream& operator << (ostream&, Polynomial&);
friend istream& operator >> (istream&, Polynomial&);

private:
        Term *termArray;
        int capacity;
        int terms;
};
#endif
```

/* hw2a 와 동일한 부분은 설명을 생략하도록 한다.
 * line 21: multiply operator overloading
*/

## ii. polyb.cpp

```
#include <iostream>
#include "polyb.h"
using namespace std;

istream& operator >> (istream& is, Polynomial& p)
{
        int noofterms;
        float coef;
        int exp;

        is >> noofterms;

        for (int i = 0; i < noofterms; i++)
        {
                is >> coef >> exp;
                p.NewTerm(coef, exp);
        }
        return is;
}

ostream& operator << (ostream& os, Polynomial& p)
{
        for (int i = 0; i < p.terms; i++)
        {
                if (p.termArray[i].coef > 0)
                {
                        if (i != 0)
                        {
                                if (p.termArray[i].coef !=
                                    ↪ 1)
                                        os << "+" << p.
                                            ↪ termArray
                                            ↪ [i].coef;
                                else
                                        os << "+";
                        }
                        else
                                if (p.termArray[i].coef !=
                                    ↪ 1)
                                        os << p.
                                            ↪ termArray
                                            ↪ [i].coef;
                }

                if (p.termArray[i].coef < 0)
                {
                        if (p.termArray[i].coef != -1)
                                os << p.termArray[i].
                                    ↪ coef;
                        else
                                os << "-";
                }

                if (p.termArray[i].exp != 0)
                        os << "x^" << p.termArray[i].exp
                            ↪ ;
        }

        for (int i = 0; i < p.terms; i++)
                for (int j = i + 1; j < p.terms; j++)
                {
                        if (p.termArray[i].exp < p.
                            ↪ termArray[j].exp)
                        {
```

```
56                                  Term temp = p.
                                        ↪ termArray[i];
57                                  p.termArray[i] = p.
                                        ↪ termArray[j];
58                                  p.termArray[j] = temp;
59                              }
60                      }
61
62          os << endl;
63          return os;
64  }
65
66  Polynomial::Polynomial():capacity(1), terms(0)
67  {
68          termArray = new Term[capacity];
69  }
70
71  void Polynomial::NewTerm(const float theCoef, const int
        ↪ theExp) // theCoeff -> theCoef
72  {
73          if (terms == capacity)
74          {
75                  capacity *= 2;
76                  Term * temp = new Term [capacity];
77                  copy(termArray, termArray + terms,
                            ↪ temp);
78                  delete [] termArray;
79                  termArray = temp;
80          }
81
82          termArray[terms].coef = theCoef;
83          termArray[terms++].exp = theExp;
84  }
85
86  Polynomial Polynomial::operator + (Polynomial& b)
87  {
88          Polynomial c;
89          int aPos = 0, bPos = 0;
90
91          while ((aPos < terms) && (bPos < b.terms))
92          {
93                  if (termArray[aPos].exp == b.termArray[
                            ↪ bPos].exp)
94                  {
95                          float t = termArray[aPos].coef +
                                    ↪ b.termArray[bPos].coef;
96                          if (t)
97                                  c.NewTerm(t, termArray
                                            ↪ [aPos].exp);
98                          aPos++;
99                          bPos++;
100                 }
101                 else if (termArray[aPos].exp < b.
                            ↪ termArray[aPos].exp)
102                 {
103                         c.NewTerm(b.termArray[bPos].
                                    ↪ coef, b.termArray[bPos
                                    ↪ ].exp);
104                         bPos++;
105                 }
106                 else
107                 {
108                         c.NewTerm(termArray[aPos].
                                    ↪ coef, termArray[aPos].
                                    ↪ exp);
109                         aPos++;
110                 }
111         }
112
113         for (; aPos < terms; aPos++)
114                 c.NewTerm(termArray[aPos].coef,
                            ↪ termArray[aPos].exp);
115
116         for (; bPos < terms; bPos++)
117                 c.NewTerm(b.termArray[bPos].coef, b.
                            ↪ termArray[bPos].exp);
118
119         return c;
120 }
121
122 Polynomial Polynomial::operator * (Polynomial& b)
123 {
124         Polynomial c;
125         int aPos = 0, bPos = 0;
126
127         for (aPos = 0; aPos < terms; aPos++)
128         {
129                 for (bPos = 0; bPos < b.terms; bPos++)
130                 {
131                         int tempCoef = termArray[aPos].
                                    ↪ coef * b.termArray[
                                    ↪ bPos].coef;
132                         int tempExp = termArray[aPos].
                                    ↪ exp + b.termArray[
                                    ↪ bPos].exp;
133
134                         c.NewTerm(tempCoef, tempExp)
                                    ↪ ;
135                 }
136         }
137
138         for (int i = 0; i < c.terms; i++)
139         {
140                 for (int j = i + 1; j < c.terms; j++)
141                 {
142                         if (c.termArray[i].exp == c.
                                    ↪ termArray[j].exp)
143                         {
144                                 c.termArray[i].coef += c.
                                            ↪ termArray[j].
                                            ↪ coef;
145                                 c.termArray[j] = c.
```

```
                                        ↪ termArray[c.
                                        ↪ terms - 1];
146                                 c.terms--;
147                         }
148                 }
149         }
150
151         for (int i = 0; i < c.terms; i++) // 내림차순 정렬
152                 for (int j = i + 1; j < c.terms; j++)
153                         if (c.termArray[i].exp < c.
                                ↪ termArray[j].exp)
154                         {
155                                 Term temp = c.
                                        ↪ termArray[i];
156                                 c.termArray[i] = c.
                                        ↪ termArray[j];
157                                 c.termArray[j] = temp;
158                         }
159
160         return c;
161 }
```

/* hw2a 와 동일한 부분은 설명을 생략하도록 한다.

* line 122: *this 와 b 를 곱한 결과를 반환한다.
* line 138 - 149: 곱셈을 통해 같은 차수의 항이 발생하는 경우를 처리하기 위하여, 각 항들을 하나의 항으로 합친 뒤 연산 후 더 이상 필요하지 않은 항들을 제거하는 코드를 다항식 곱셈 함수에 다시 작성하였다.

*/

## iii. hw2b.cpp

```
1  #include <iostream>
2  using namespace std;
3  #include "polyb.h"
4
5  int main()
6  {
7          Polynomial p1, p2;
8
9          cin >> p1 >> p2;
10         Polynomial p3 = p1 * p2;
11         cout << p1 << p2 << p3;
12 }
```

/* hw2a 와 동일한 부분은 설명을 생략하도록 한다.

* line 10 - 11: 다항식 곱셈 연산 결과를 Polynomial 타입의 p3 에 새로이 저장 및 각 다항식들을 출력한다.

*/

## iv. results

### iv.1 makefile

```
1  hw2b:hw2b.o polyb.o
2          g++ -o hw2b hw2b.o polyb.o
3  hw2b.o polyb.o:polyb.h
```

### iv.2 poly.in

```
1  3 3.0 5 2.0 3 -4.0 0
2  4 1.0 8 -7.0 5 -1.0 3 -3.0 0
```

### iv.3 compile

```
1  [B711222@localhost hw2d]$ ./hw2b < poly.in
2  3x^5+2x^3-4
3  x^8-7x^5-x^3-3
4  3x^13+2x^11-21x^10-21x^8-2x^6+19x^5-2x^3+12
```