**자료구조 01**

****

**중앙대학교**

**자료구조 05 분반**

**20165079 김영빈**

**20160125 김소령**

**목차**

과제

[**1.** **과제 협업 방식** 3](#_Toc67793957)

[**2.** **다항식** 3](#_Toc67793958)

[**다항식 방식 1 - C언어** (파일명 : assignment\_01\_polynomial\_01.c) 3](#_Toc67793959)

[**다항식 방식 1 – Python** (파일명 : assignment\_01\_polynomial\_01.py) 4](#_Toc67793960)

[**다항식 방식 2 - C언어** (파일명 : assignment\_01\_polynomial\_02.c) 5](#_Toc67793961)

[**다항식 방식 2 – Python** (파일명 : assignment\_01\_polynomial\_02.py) 7](#_Toc67793962)

[**3.** **행렬** 8](#_Toc67793963)

[**행렬 - C언어** 8](#_Toc67793964)

[**행렬 – Python** 9](#_Toc67793965)

[**4. 결과 화면** 10](#_Toc67793966)

[**1. 다항식** 10](#_Toc67793967)

[**2. 행렬** 14](#_Toc67793968)

# **과제 협업 방식**

C언어와 Python 중 하나를 선택해서 과제를 해도 좋다고 말씀하셨지만, 두 가지 모두 공부의 필요성을 느끼고 있었기에 언어 선택에서 고민이 있었습니다. 그러던 중, 교수님께서 두 가지 언어를 모두 설명해주시는 것에 감명을 받아, 두 가지 언어로 같은 문제를 함께 풀어보면서 공부하기로 했습니다.

각자 맡은 언어로 과제를 해결한 뒤, 코드를 공유하고 서로의 코드를 설명하고 해결 방안을 작성해보는 등의 시간을 통해 코드 리뷰를 했습니다.

* C언어 코드 작성 – 김영빈, 코드 리뷰 – 김소령
* Python 코드 작성 – 김소령, 코드 리뷰 – 김영빈

과제의 파일명은 assignment\_{문제번호}\_{문제이름}\_{필요시-인덱스-추가}.{확장자}

# **다항식**

다항식을 저장하는 2개 방식을 배웠다.

* 방식 1. 모든 차수의 계수를 저장한다.
* 방식 2. 0이 아닌 계수의 차수만 저장한다.

## **다항식 방식 1 - C언어** (파일명 : assignment\_01\_polynomial\_01.c)

C언어로 작성한 코드 내에서 선언한 변수와 함수의 내용은 다음과 같다.

Polynomial Struct

* Degree: 항의 최고 차수
* Coef[MAX\_DEGREE][2]: 계수와 차수를 저장함

Power

* Poly\_get\_value에서 쓰이는 함수로 power을 계산하는 함수

Poly\_add

* 다항식을 더해주는 함수
* 두 다항식의 덧셈 결과를 polynomial C에 저장하여 반환한다.
* 두 다항식의 차수를 각각 비교해서 차수가 높은 다항식을 먼저 C 에 넣기 시작한다.
* C에 높은 차수의 다항식을 하나씩 저장하다가 두 다항식의 차수가 같은 계수를 연산하는 시점에는 두 다항식을 더한 뒤 결과값을 C에 저장한다.
* C를 반환하면 두 다항식의 덧셈 결과가 반환된다.

Poly\_mul

* 다항식을 곱해주는 함수
* 두 다항식의 곱셈 결과를 polynomial D에 저장하여 반환한다
* 두 다항식의 차수를 각각 I 와 j 로 설정하여 결과 다항식의 알맞은 차수에 곱을 저장한다.
* D를 반환하면 두 다항식의 곱셈 결과가 반환된다.

Poly\_init

* 문자열로 입력 받은 다항식 정보를 실제 숫자로 변환하여 다항식에 저장하는 initialize 함수
* Strtok를 이용해서 space를 token으로 하여 사용자에게 입력 받은 문자에서 각 숫자를 추출하여 저장하는 함수

Poly\_print

* 다항식을 출력해주는 함수

Poly\_get\_value

* 다항식 f(x)에 특정 x 값을 넣었을 때 결과 값을 반환해주는 함수

## **다항식 방식 1 – Python** (파일명 : assignment\_01\_polynomial\_01.py)

Python으로 작성한 코드 내에서 선언한 클래스(이하 객체)와 함수는 다음과 같다.

Class polynomial

* polynomial이라는 객체 생성을 위한 클래스 정의
* 각 항의 계수들을 담은 리스트 coef와 함께 선언됨
* Degree는 최고차항의 차수를 담는 것으로, 모든 계수가 담긴 리스트의 길이보다 1 작다.
* Print\_poly() : 반복문을 돌며 계수를 차례로 출력한다.
* Substitute(x) : x로 전달받은 수를 각 계수들과 곱해서 더한 후 반환한다.

Poly\_add

* 두 개의 계수 리스트를 받아서 같은 차수끼리 더해준다.
* 이 때, 각 리스트를 돌아다니기 위해 pos\_i 라는 변수를 사용했다.
* 더 큰 차수를 먼저 확인하고, 같은 차수는 계수끼리 더하는 방식이다.

Poly\_multiply

* 두 개의 계수 리스트를 받아서 각 항들을 모두 곱한다.
* 자리수를 맞추기 위해 전체 자리수를 구하고, 앞 뒤로 0을 채운다.
* 반복문을 너무 많이 사용하여 가독성이 떨어지고, 효율이 좋지 않을 수 있겠다는 생각을 했다.

Get\_equation

* 띄어쓰기를 기준으로 수를 구분하여 정수형으로 리스트에 저장한다.
* 최종적으로 이상에 만든 리스트로 polynomial 객체를 반환한다.

## **다항식 방식 2 - C언어** (파일명 : assignment\_01\_polynomial\_02.c)

C언어로 작성한 코드 내에서 선언한 변수와 함수의 내용은 다음과 같다.

Polynomial Struct

* Amount: 항의 개수를 저장함
* Coef[MAX\_DEGREE][2]: 계수와 차수를 저장함

Power

* Poly\_get\_value에서 쓰이는 함수로 power을 계산하는 함수

Poly\_add

* 다항식을 더해주는 함수
* 두 다항식의 덧셈 결과를 polynomial C에 저장한다. 이때 C 구조체는 포인터 주소를 함수의 매개변수로 받아왔기 때문에 별 다른 return을 하지 않아도 된다.
* 모든 차수에 대한 계수를 저장하는 것이 아니기에 항의 개수를 이용해서 덧셈을 해야 한다. 두 다항식을 더했을 때 최대로 나올 수 있는 항의 개수는 더하는 두 다항식의 항의 개수를 더한 것과 같기 때문에 처음에는 C의 항의 개수를 A와 B의 항의 개수를 더 한 것으로 한다. 따라서 처음에 두 다항식의 차수가 같다면 C의 항의 개수를 1개씩 줄여주면서 최종 결과물로 나오는 항의 개수를 미리 예측한다.
* 덧셈을 할 때에는 두 다항식을 저장하고 있는 배열에서 두 다항식의 차수를 서로 비교하면서 더 큰 차수를 먼저 결과 다항식인 C에 더해준다. 만약 같은 차수의 다항식이 나온다면 두 계수를 더해서 저장한다.

Poly\_mul

* 다항식을 곱해주는 함수
* 두 다항식을 곱해서 D 구조체에 저장한다. 이때 D 구조체는 포인터 주소를 함수의 매개변수로 받아왔기 때문에 별 다른 return을 하지 않아도 된다.
* 구조체 D의 최대 항의 수는 A와 B의 항의 수를 곱한 것이다. D의 항의 개수를 일단 A와 B 항의 수의 곱으로 설정한다.
* 우선 temp라는 2차 포인터에 D의 항의 수만큼 동적 할당을 해서 두 다항식의 곱셈 결과를 저장하도록 한다.
* 이때 두 다항식의 곱셈 결과에서 중복된 차수는 더해서 정렬해야 하기 때문에 already\_in이라는 변수를 이용해서 곱셈 결과에 존재하는 모든 차수를 저장한다. 예를 들어 곱셈 결과가 {{2, 3}, {2, 2}, {2, 1}, {2, 2}, {2, 0}} 이라면 already\_in이라는 already\_in 배열에는 3, 2, 1, 0이 들어가게 된다.
* 이후에 already\_in의 변수에서 차수를 하나씩 꺼내면서 해당 차수와 같은 차수를 가진 계수를 temp배열에서 찾아서 더한 결과를 D에 넣으면서 곱셈 결과를 만든다.

Poly\_init

* 문자열로 입력 받은 다항식 정보를 실제 숫자로 변환하여 다항식에 저장하는 initialize 함수
* Strtok를 이용해서 space를 token으로 하여 사용자에게 입력 받은 문자에서 각 숫자를 추출하여 저장하는 함수
* 이때 strtok에서 while문을 두 번 사용하는 이유는 연속으로 추출되는 값을 각 row의 0번과 1번 인덱스에 저장하기 위해서이다.

Poly\_print

* 다항식을 출력해주는 함수

Poly\_get\_value

다항식 f(x)에 특정 x 값을 넣었을 때 결과 값을 반환해주는 함수

## **다항식 방식 2 – Python** (파일명 : assignment\_01\_polynomial\_02.py)

Class polynomial

* polynomial이라는 객체 생성을 위한 클래스 정의
* 계수가 0이 아닌 항들의 차수와 계수를 담은 리스트 coef와 함께 선언됨
* Num은 0이 아닌 항의 개수이다.
* Print\_poly() : 반복문을 돌며 차수와 계수를 차례로 출력한다.
* Substitute(x) : x로 전달받은 수를 각 계수들과 곱해서 더한 후 반환한다.

Poly\_add

* 두 개의 차수-계수 리스트를 받아서 같은 차수끼리 더해준다.
* 이 때, 각 리스트를 돌아다니기 위해 pos\_i 라는 변수를 사용했다.
* 더 큰 차수를 먼저 확인하고, 같은 차수는 계수끼리 더하는 방식이다.
* 더 큰 차수를 먼저 확인하기 때문에 자동으로 정렬이 된다.

Poly\_multiply

* 모든 항들은 서로 곱해지는데, 그 결과의 차수가 같은 경우에는 더한다.
* 이를 차수를 key, 계수를 value로 하는 dictionary에 담은 후 차수를 기준으로 정렬하여 result에 담아 객체 polynomial를 만들어 반환한다.

Get\_equation

* 띄어쓰기를 기준으로 수를 구분하여 정수형으로 리스트에 저장한다.
* 최종적으로 이상에 만든 리스트로 polynomial 객체를 반환한다.

# **행렬**

행렬을 저장하는 방식 2가지를 배웠다.

* 방식 1: 배열에 그대로 저장하는 방법
* 방식 2: 0이 아닌 값만 저장하는 방법

## **행렬 - C언어** (파일명 : assignment\_02\_matrix.c)

C언어의 경우 선언한 변수와 함수의 내용은 다음과 같다.

Add\_normal

* 2차원 배열 C를 선언한다. 두 행렬의 덧셈 결과를 저장하기 위함이므로 두 행렬의 크기만큼 C를 동적 할당 한다.
* 두 행렬을 인덱스에 맞춰 더하고 저장한다.
* 두 행렬을 더한 뒤 출력하고 할당 받은 메모리를 free 해준다.

Multiply\_normal

* 2차원 배열 D를 선언한다. 두 행렬의 곱셈 결과를 저장하기 위함이므로 두 행렬의 크기만큼 D를 동적 할당 한다.
* 두 행렬을 인덱스에 맞춰 곱하고 저장한다.
* 두 행렬을 곱한 뒤 출력하고 할당 받은 메모리를 free 해준다.

Add\_sparse

* 2차원 배열 C를 선언해서 두 행렬의 덧셈 결과값을 저장한다. 이때 C는 2차 포인터 형으로 선언한다. 왜냐하면 입력받는 A와 B의 row 값이 달라질 수 있기 떄문이다.
* 두 행렬 중 행과 열의 값이 가장 적은 값부터 C에 저장되게 하기 위해서 while 문 안에 2중 if 문으로 행과 열의 값을 비교하여 적은 값부터 C에 저장한다.
* 만약 행과 열이 같은 값이 있다면 두 값을 더해서 C에 저장한다.
* 두 행렬을 더한 뒤 출력하고 할당 받은 메모리를 free 해준다.

Multiply\_sparse

* 2차원 배열 D를 선언해서 두 행렬의 곱셈 결과값을 저장한다. 이때 D는 2차 포인터 형으로 선언한다. 왜냐하면 입력받는 A와 B의 row 값이 달라질 수 있기 때문이다.
* 두 행렬의 곱셈의 경우에는 인덱스의 조작이 매우 중요하다. (0,0)위치의 숫자부터 (matrix\_space, matrix\_space)위치의 숫자까지 해당 인덱스에 곱셈의 결과가 존재하는지 여부를 파악하여 곱셈의 결과를 저장한다. 이때 matrix\_space는 사용자가 입력한 행렬의 규격이다.
* 두 행렬을 곱한 뒤 출력하고 할당 받은 메모리를 free 해준다.

## **행렬 – Python** (파일명 : assignment\_02\_matrix.py)

Class matrix

* 행렬이 되는 객체
* N\*N의 규격을 가진 행렬인 경우 size에는 N의 제곱이 아닌 N을 담는다.
* 행렬의 내용을 담은 data도 함께 선언된다.
* Print\_normal() : 행렬의 모든 수들이 담긴 normal의 경우 반복문을 돌면서 그 내용을 출력한다.
* Print\_sparse() : 0이 아닌 수의 내용들만 담긴, 희소행렬 sparse의 경우에는 리스트 속 리스트의 형식으로 들어가기 때문에 반복문의 내용이 조금 다르다.

Get\_normal

* 행렬의 내용들을 띄어쓰기로 구분해서 숫자만 담긴 리스트를 만들어 반환한다.

Add\_normal

* 같은 자리에 있는 수들끼리 더한 뒤 리스트로 반환한다.

Multiply\_normal

* 행렬의 곱셈 방식에 의해 다중 반복문을 돌려 리스트를 만들어 반환한다.

Get\_sparse

* 단순히 숫자들만 담기는 것이 아니라, 위치와 내용이 함께 담긴 리스트들이 담긴다.
* Get\_normal에서 받아왔던 행렬을 이용해서 다시 저장한다. 예시에서 행렬의 내용을 두 번 입력받지는 않기 때문이다.
* 이 때 index의 내용을 행렬의 규격 s로 나눈 몫이 행번호, 나머지가 열번호가 된다.

Compare\_same

* 아래 add\_sparse에서 차수끼리 비교하는 일이 중첩되어서 함수로 만들어 빼두었다.
* 같은 일을 반복하기에 재귀함수를 넣어두었다.

Add\_sparse

* 위치가 온전히 일치하는 경우에는 서로를 더하고, 그렇지 않은 경우에는 더 앞의 위치에 있는 수를 먼저 적는다.

Multiply\_sparse

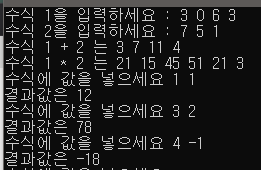
* 행렬의 곱을 살펴보면 행렬 m1, m2에 대해서 결과 m의 (i, j)에 들어갈 내용은 m1(i, k) \* m2(k, j)의 값이다.
* (I, j)에 들어갈 내용을 찾기 위해서 앞 행렬의 행번호가 i인지, 동시에 뒤 행렬의 열번호가 j인 것이 있는지, 또한 각각의 열번호와 행번호가 일치하는지를 확인하여 곱셈을 이어가는 것이다.

# **4. 결과 화면**

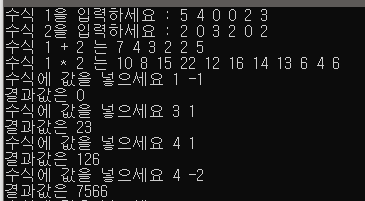
## **1. 다항식**

**[ 방식 1 - C언어 ]**

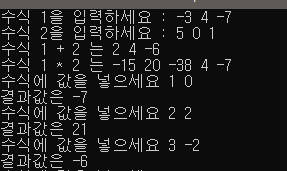
1)



2)

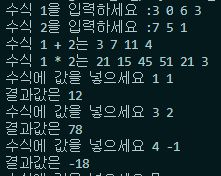


3)

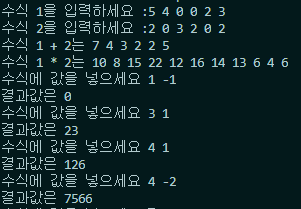


**[ 방식 1 – Python ]**

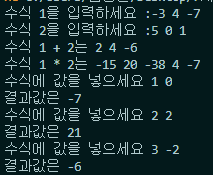
1)



2)

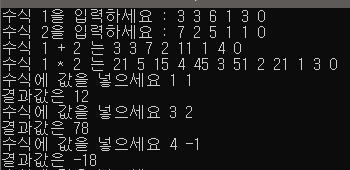


3)

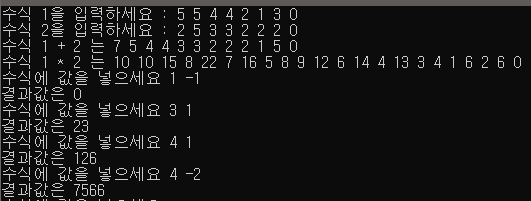


**[ 방식 2 - C언어 ]**

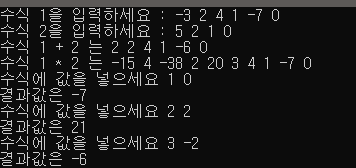
1)



2)

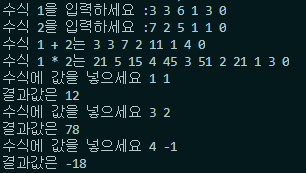


3)

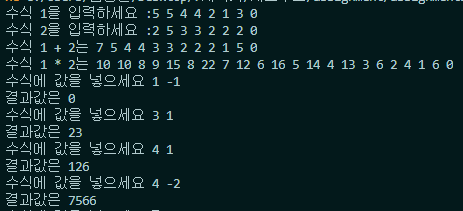


**[ 방식 1 – Python ]**

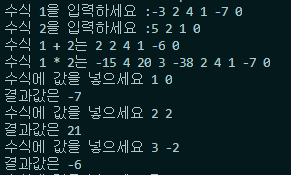
1)



2)



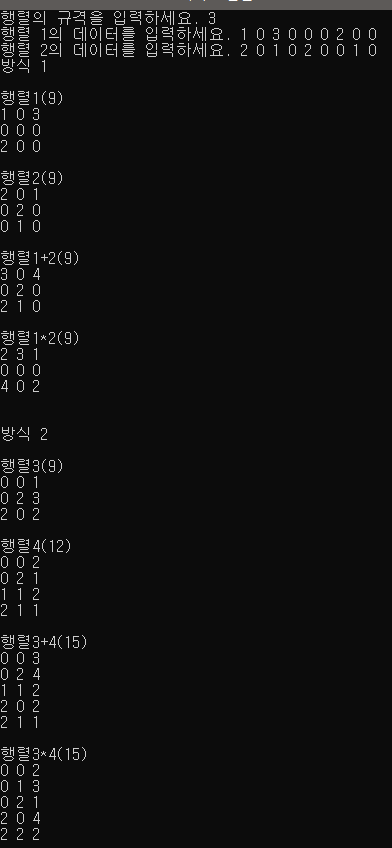
3)

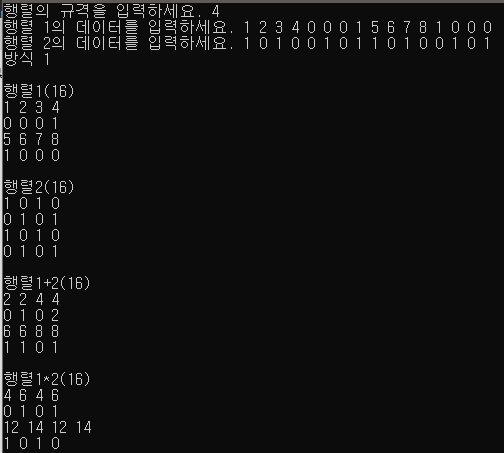


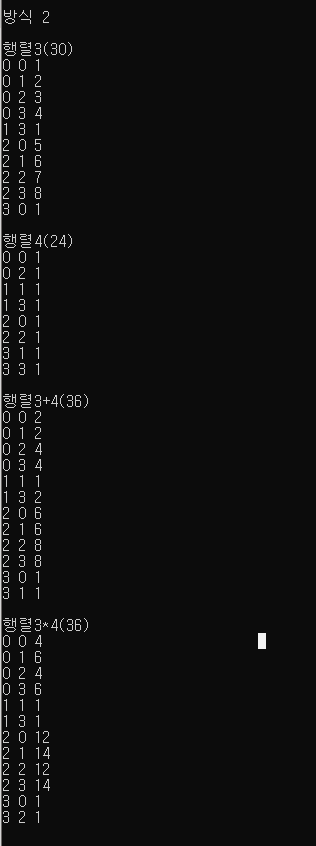
## **2. 행렬**

**C언어**

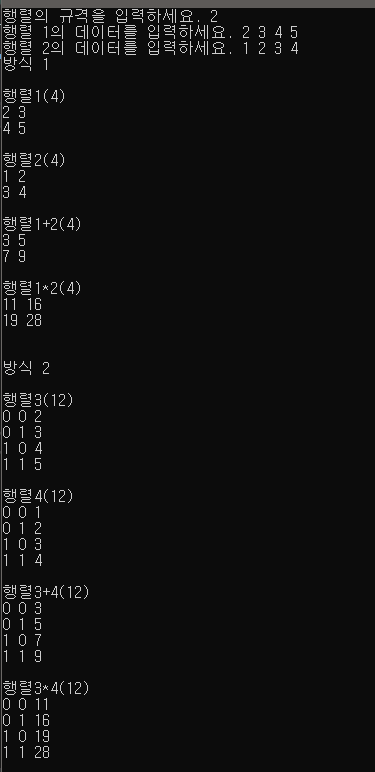
1)



2)

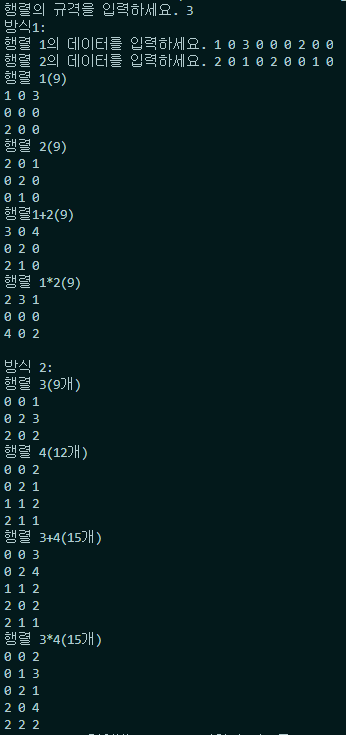


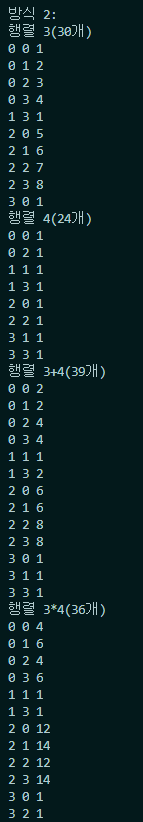
3)

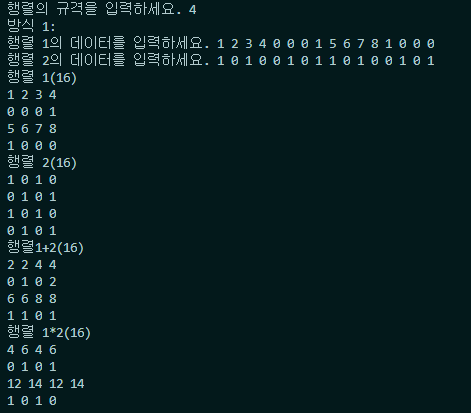


**[ 방식 1 - Python ]**

1)



2)



3)

