

# 과제 1

자료구조 만들기

파이썬의 List 내부 구조 파악

다항식의 저장 방식 개선

희소 행렬 저장 방식의 곱셈

미로 표현하기

20232907 정현승

# 목차

1. 문제 1: 자료구조 만들기	3
2. 문제 1 해결 방안	3
3. 문제 1의 결과	6
4. 문제 2: 파이썬의 List 내부 구조 파악	16
5. 문제 2 파이썬 List의 내부 구조	17
6. 문제 3: 다항식의 저장 방식 개선	17
7. 문제 3 해결 방안	17
8. 문제 3의 결과	19
9. 문제 4: 희소 행렬 저장 방식의 곱셈	21
10. 문제 4 해결 방안	21
11. 문제 4의 결과	32
12. 문제 5: 미로 저장하기	34
13. 문제 5 해결 방안	34
14. 소스코드	35
15. 참고문헌	36

## 1. 문제 1: 자료구조 만들기

포인터 변수 하나만을 이용해 정수형, 실수형, 문자열 변수를 저장한다. 이때 정수형은 1부터 차례대로 채우고, 실수형은 1.0부터 0.1씩 더해 차례대로 저장하며, 문자열은 hi1, hi2, ... 의 순서대로 저장한다. a, b, c를 입력받고 정수형 변수의 저장 공간은  $a \times b$ 개, 실수형 변수의 저장 공간은  $b \times c$ 개, 문자열의 저장 공간은  $c \times a$ 개 저장한다. a, b, c는 1 이상 9 이하의 숫자이며, 모든 공간은 동적 할당받아 저장한다.

## 2. 문제 1 해결 방안

문제에서 포인터 변수 ptr은 정수(int)형 2차원 배열, 실수(double)형 2차원 배열, 문자열(char\*) 2차원 배열 3개를 원소로 하는 배열을 가리켜야 한다. 2차원 배열 3개를 모아 놓은 배열이니 이 배열을 가리키는 포인터 변수는 3차원 포인터가 되어야 하며, 각 2차원 배열 당 저장하는 자료형이 다르므로 ptr의 자료형은 void\*\*\*이 된다. 어차피 형 변환을 해 주어야 하는 void 포인터의 특성상 void\*나 void\*\*로도 선언이 가능하나, 동적 할당으로 배열의 공간을 만들 때만큼은 형 변환을 피하고자 3차원 포인터로 선언하였다.

동적 할당은 4번에 나누어 받았다. 1번만 할당받아 이 공간을 크기에 맞게 사용해도 되지만 보는 사람 입장에서 코드 이해가 어려워질 것을, 그리고 설계가 약간 더 복잡해진다는 점을 고려했고, 그렇다고 동적 할당을 자주 받으면 그만큼 프로그램의 속도가 느려진다는 점도 고려하여, 4번에 나누어 받았다. 2차원 배열부터는 배열 각각 할당받지 않고 한 번만 할당받아 세 부분으로 나누어 사용했다. 차원이 달라질 때만 할당을 다시 받았다. 그러니까 void\*\*[3], void\*[], (int [], double[], char\*[]), char[] 이렇게 4번만 할당받고 이 공간을 적절히 나누었다.

이를 코드로 설명하면 다음과 같다. 먼저 void\*\*\* ptr에 void\*\*[3] 공간을 할당받는다. 이후 a, b, c를 입력받고, ptr[0]에 void\*[a+b+c] 공간을 할당받는다. 이때 ptr[0]은 void\*[a] 공간만 필요하고, 아직 NULL 값이 저장된 ptr[1], ptr[2]는 각각 void\*[b], void\*[c]의 공간이 필요하다. 따라서 한 번에 많이 할당받고 남은 ptr[0]의 공간을 ptr[1], ptr[2]에 각각 나눈다. ptr[1]에는 ptr[0]+a 값을 넣는데, ptr[0]이 a만큼의 포인터 저장 공간만이 필요하기에 그 뒤의 남은 공간을 ptr[1] 쓰라고 넘겨주는 것이다. 같은 방식으로 ptr[2]에도 ptr[1]+b 값을 넣어준다. 그러니까 ptr[0][a]==ptr[1][0] 이고, ptr[1][b]==ptr[2][0] 이 되는 구조이다. 이를 a=2, b=3, c=2일 때 그림으로 표현하면 그림 2.1과 같다. ptr[0] 2차원 배열의 공간, ptr[1] 2차원 배열의 공간, ptr[2] 2차원 배열의 공간 전부 연속되어 있으며, 따라서 ptr[0][2]==ptr[1][0] 이고, ptr[1][3]==ptr[2][0] 이다.

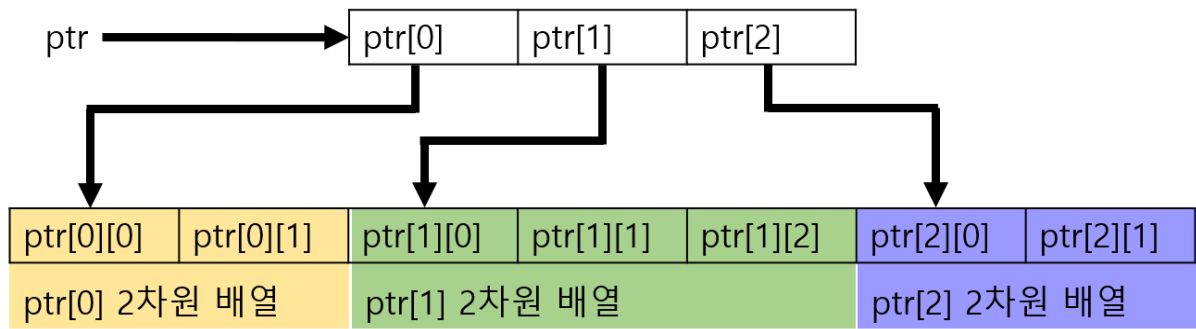


그림 2.1

이로써 현재 상황은 다음과 같다.

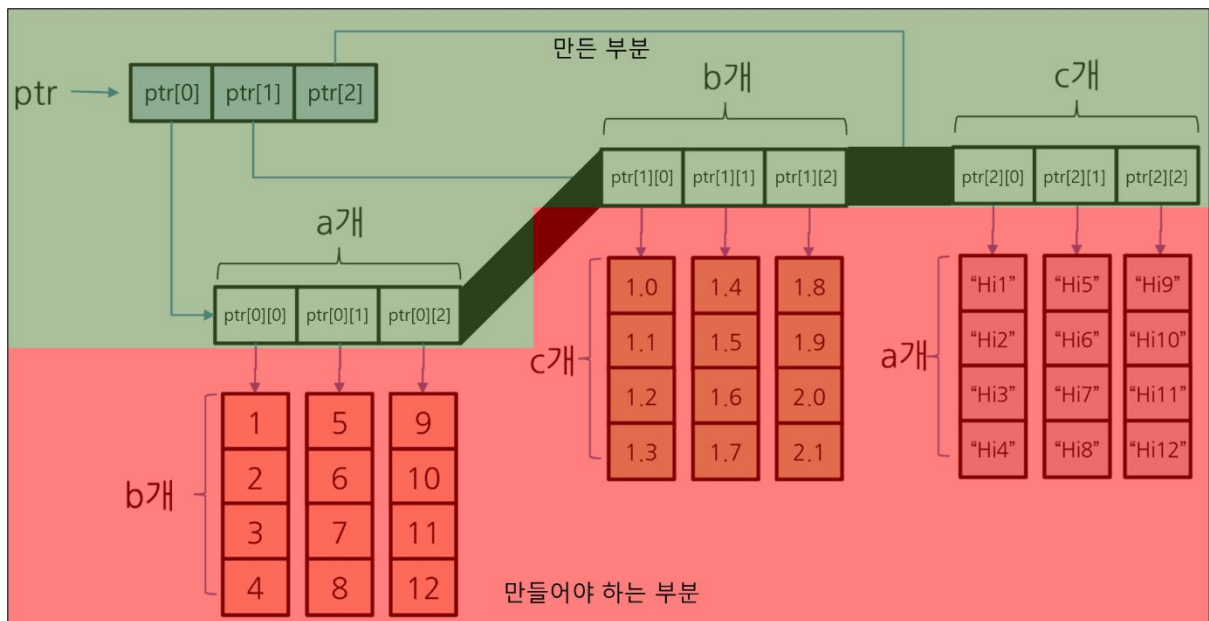


그림 2.2

다음으로 2차원 배열에 들어갈 1차원 배열을 만들어 줄 차례이다. b개짜리의 int형 1차원 배열이 a개, c개짜리의 double형 1차원 배열이 b개, a개짜리의 문자열 1차원 배열이 c개 필요하다. 이 전체 메모리를 1번만 할당받아 알맞게 나누어 배분하였다. 먼저 길이  $(a*b*\text{sizeof}(\text{int})) + (b*c*\text{sizeof}(\text{double})) + (c*a*\text{sizeof}(\text{char}))$ 만큼의 용량을 동적 할당받았다. 이 동적 할당받은 주소를 ptr[0][0]에 넣었다. 이후 int형 변수가 b개 들어갈 만큼의 공간을 남겨 두고 ptr[0][1]에 할당했다. 이를 ptr[0][a]에 주소 값이 할당될 때까지 반복한다. 이때 int형 2차원 배열의 크기는  $a \times b$ 로, ptr[0][a]는 int형 2차원 배열에 존재하지 않는 범위이다. 따라서 원래라면 할당받지 않은 공간에 접근하였으므로 에러가 발생하여야 한다. 그러나, 우리는 2차원 배열의 공간도 연속으로 할당했었다. 따라서  $\text{ptr}[0][a] = \text{ptr}[1][0]$  이 된다. ptr[1][0]은 우리가 한꺼번에 할당받은 공간이므로 전혀 에러가 나지 않는다. 오히려 int형 2차원 배열이 쓰고 남은 공간을 double형 2차원 배열이 쓰라고 넘겨준 상황이라고 해석할 수 있다. 즉, 단순히 1차원 배열로 사용할 공간을 크기에 맞게 나누

어 저장한 것뿐임에도 2차원 배열이 완성되었으며, 이에 더해 만들고 남은 공간은 다음 2차원 배열이 쓰도록 넘겨준 것이다! double형 2차원 배열도 똑같이 반복하여 2차원 배열을 만들고, 남은 공간은 문자열 2차원 배열이 쓰도록 넘겨준다. 문자열(char\*) 2차원 배열도 이를 반복하되, 공간을 넘겨줄 다른 배열이 없으므로 ptr[2][c]까지가 아니라 ptr[2][c-1]까지 이를 반복한다. 이렇게 되면 동적 할당은 총 3번만 받고 2차원 배열 3개가 만들어졌다. 이를 설명하기 위한 그림이 별도로 있으면 좋겠지만, 리본, 반죽, 아니면 그냥 줄을 팔 등에 여러 번 감는 사진을 찾기 어려워 그림으로 설명하지는 못했다.

int형 2차원 배열과 double형 2차원 배열은 이 자체로 완성되었으므로 값을 넣어준다. 이때 앞에서 2차원 배열을 연속으로 받았다는 점을 고려하여 for문 1번만으로 모든 위치에 값을 넣을 수 있다. 그러니까 &ptr[0][0][b]==ptr[0][1][0]이므로(형 변환은 생략하였다.) 2차원 배열에 for문을 2번 돌리는 게 아니라 &ptr[0][0][0] 부터 &ptr[0][0][a\*b-1]까지 각각 1부터 수를 넣음으로써 2차원 배열 전체에 값을 넣을 수 있다. 이는 double형 2차원 배열도 마찬가지로 진행할 수 있다.

아직 문자열은 포인터만 잡아놓고 실제 문자를 만들지 않았다. 여기서 마지막으로 동적 할당을 받는다. 어차피 저장할 문자열이 "Hi"와 최대 2자리의 양의 정수이므로(a, b, c가 최대 9이므로, 최대 81까지만 나온다), 각 문자열 당 널 문자를 포함하여 5바이트의 공간을 할당받았다. 한 번에 충분한 공간을 할당받고, int형과 double형 2차원 배열 채우듯 이 공간을 문자열 2차원 배열 전체에 나누었다. 그리고 각 공간에 문자열을 채워 넣었는데, sprintf 함수를 이용했다. 이 부분의 코드를 보면 char 3차원 포인터나 2차원 포인터에서 char\* \*, char\* \*\*같이 띄어쓰기가 들어가 있는데, 이는 void\*\*\*에서 void 부분이기 때문이다. 문자열의 포인터는 char형의 포인터로 바꾸면 차원이 하나 추가된다. 따라서 갑자기 형 변환으로 차원이 하나 늘어나 코드를 읽는 입장에서 혼동이 올 수 있다. 이때의 혼동을 막기 위해 (char\*)를 하나의 자료형처럼 취급한다는 것을 보여주기 위함이다. 이외에도 1차원 배열을 할당받을 때 char\* 대신 char[5]만큼의 공간을 받아 공간도 줄이고 동적 할당받는 회수 1회를 줄일 수 있었으나, 확장성과 코드를 읽는 입장에서의 이해를 위해 그렇게 하지 않았다.

이후 모든 공간이 완성되었으므로 출력을 진행하였다. 출력은 값을 넣을 때와는 달리 줄 변경이 필요하므로 평범한 2차원 배열처럼 for문 2회를 이용했다.

마지막으로 동적할당을 해제하여야 한다. 동적할당은 총 4번을 받았는데 calloc 함수가 반환한 주소는 전부 index가 0인 지점에 들어갔다는 공통점이 있다. 이 때문에 동적할당 해제 시에도 복잡한 계산 필요 없이 (char\* \*)(ptr[2][0])[0] (문자열), ptr[0][0], ptr[0], ptr 이렇게만 해제해 주면 된다.

### 3. 문제 1의 결과

문제를 동적 할당 4번만으로 해결했는데, 이 할당과 배열 생성이 정상적으로 되었는지부터 테스트해 보았다. 그림 3.1은 a=2, b=3, c=2를 입력하고 디버그 모드로 실행한 결과이다.

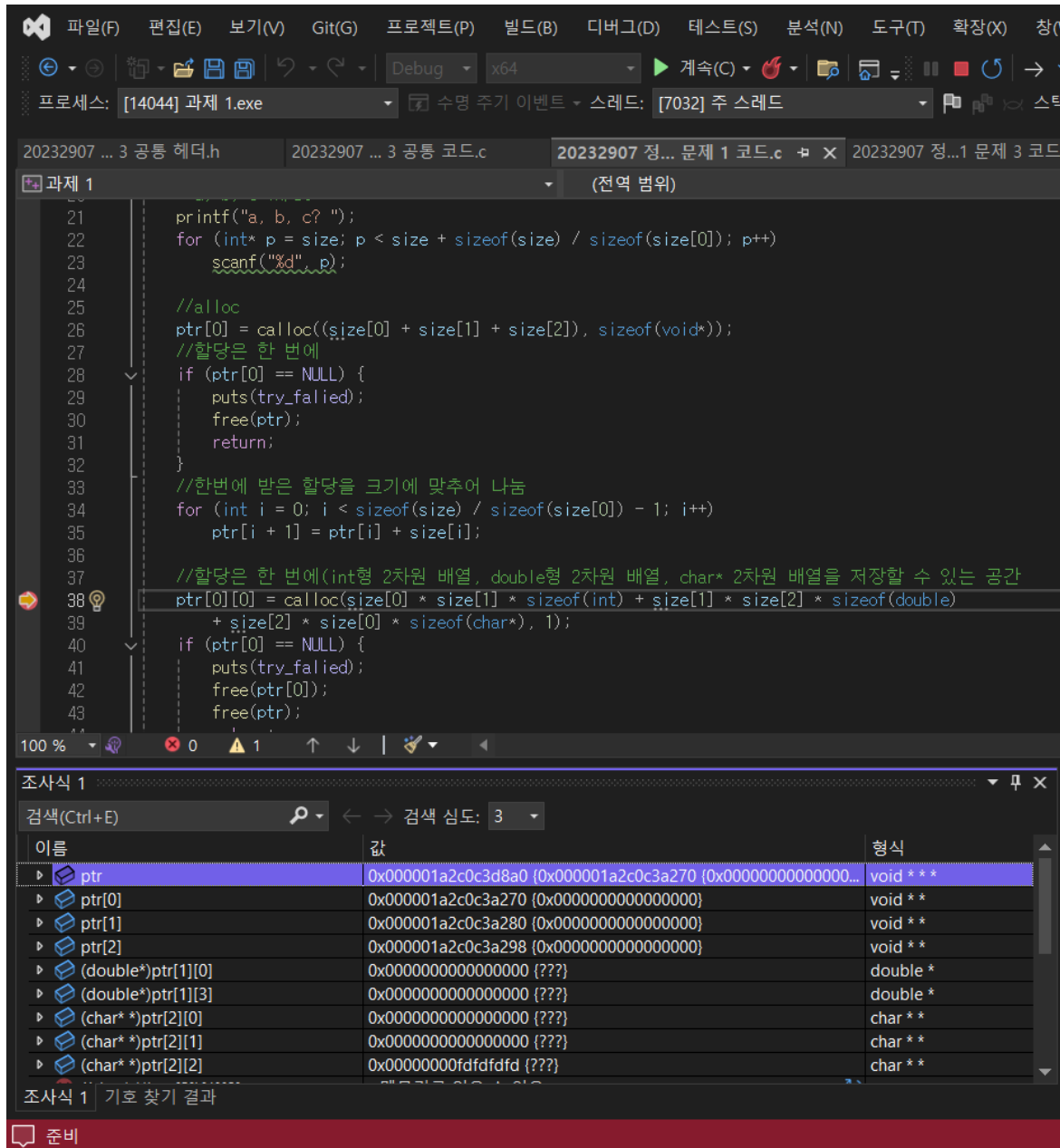


그림 3.1

위 그림 3.1은 3차원과 2차원 배열을 할당받고 각 길이에 알맞게 나눈 결과이다. 하단 "조사식1" 창을 보면, ptr[0]의 값과 ptr[1]의 값 차이가 정확히 포인터 2개 들어갈 만한 공간(0x10)이, ptr[1]의 값과 ptr[2]의 값 차이가 정확히 포인터 3개 들어갈 만한 공간(0x18)이 나온다는 점을 통해 할당 받은 공간이 성공적으로 각 길이에 나누어졌다는 점을 알 수 있다.

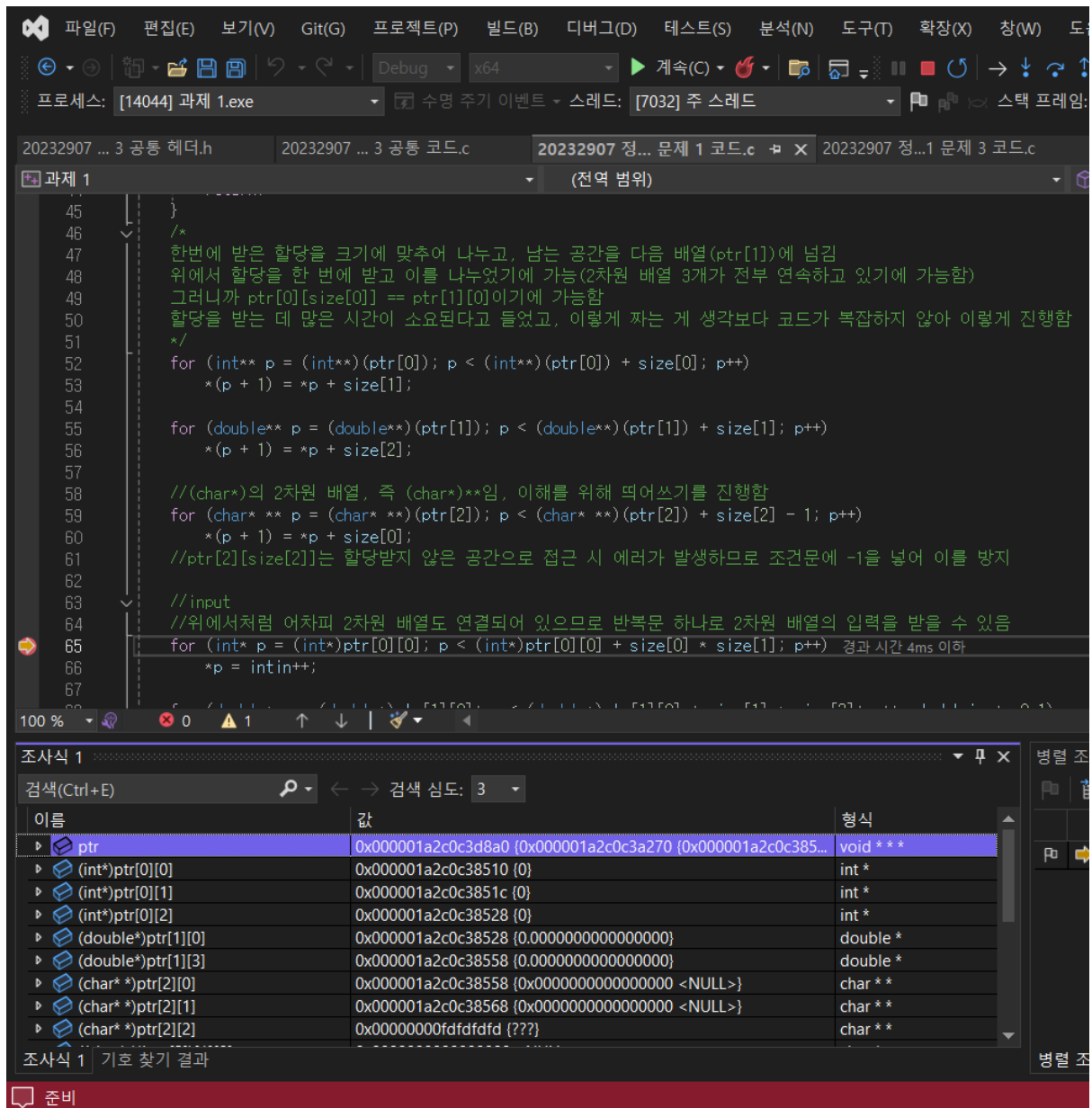


그림 3.2

이러 1차원 배열도 할당받았다. ptr[0][0]과 ptr[0][1]의 값 차이와 ptr[0][1]와 ptr[0][2]의 값 차이가 정확히 int형 변수 3개가 들어갈 공간이(0x0C), ptr[1][0]과 ptr[1][3]의 값 차이가 정확히 double형 변수 3x2=6개가 들어갈 공간이(0x30), ptr[2][0]과 ptr[2][1]의 값 차이가 정확히 char\*형 변수 2개가 들어갈 공간(0x10)이 나온다는 점, 그리고 ptr[0][2]와 ptr[1][0]에 저장된 주소값이 같고, ptr[1][3]과 ptr[2][0]에 저장된 주소값이 같다는 점을 통해 의도대로 동작하였음을 알 수 있다. 또 동적으로 할당받은 영역이 아닌 ptr[2][2]는 아무런 값도 대입하지 않아 쓰레기 값이 들어 있다는 점도 파악할 수 있다.

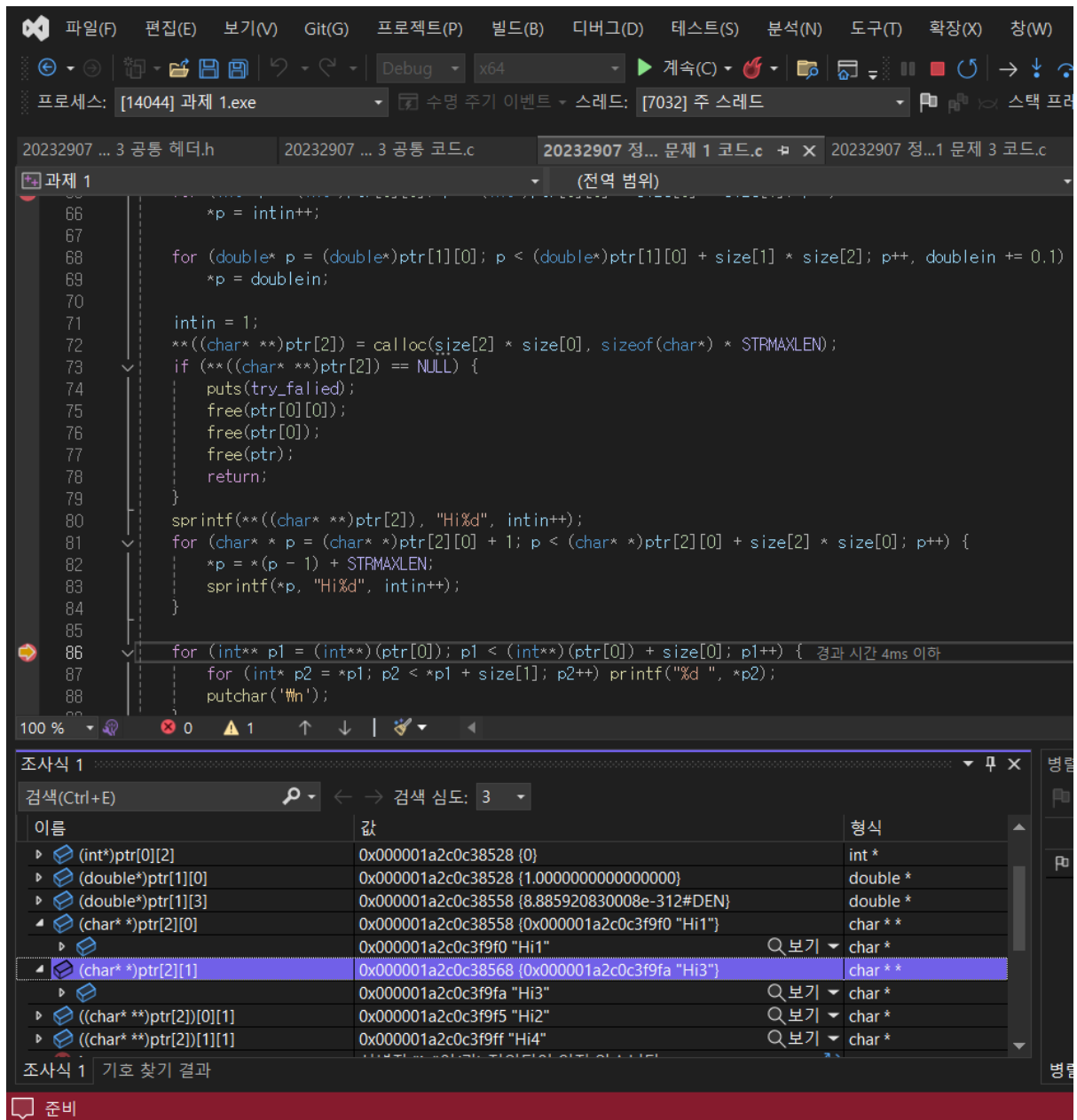


그림 3.3

이러 int, double형 변수는 물론 문자열까지 받은 후의 모습이다. "조사식1" 란에 식이 배치된 순서가 일정하지는 않은데, 문자열도 위의 그림 3.1, 그림 3.2처럼 정상적으로 할당되고 입력되었음을 알 수 있다.





그림 3.4

그림 3.4는 그림 3.3 이후 계속 실행하려 출력한 값을 찍은 것이다. 요구사항의 결과와 완전히 같은 결과가 나왔다.

이제 프로그램 종료 전 할당된 공간을 해제하는 작업만이 남았다. 그림 3.5~3.9는 할당된 공간을 해제하는 것을 하나하나 스크린샷으로 담은 것이다. "조사식1" 란을 확인함으로써 동적할당이 해제되는 과정을 볼 수 있다.

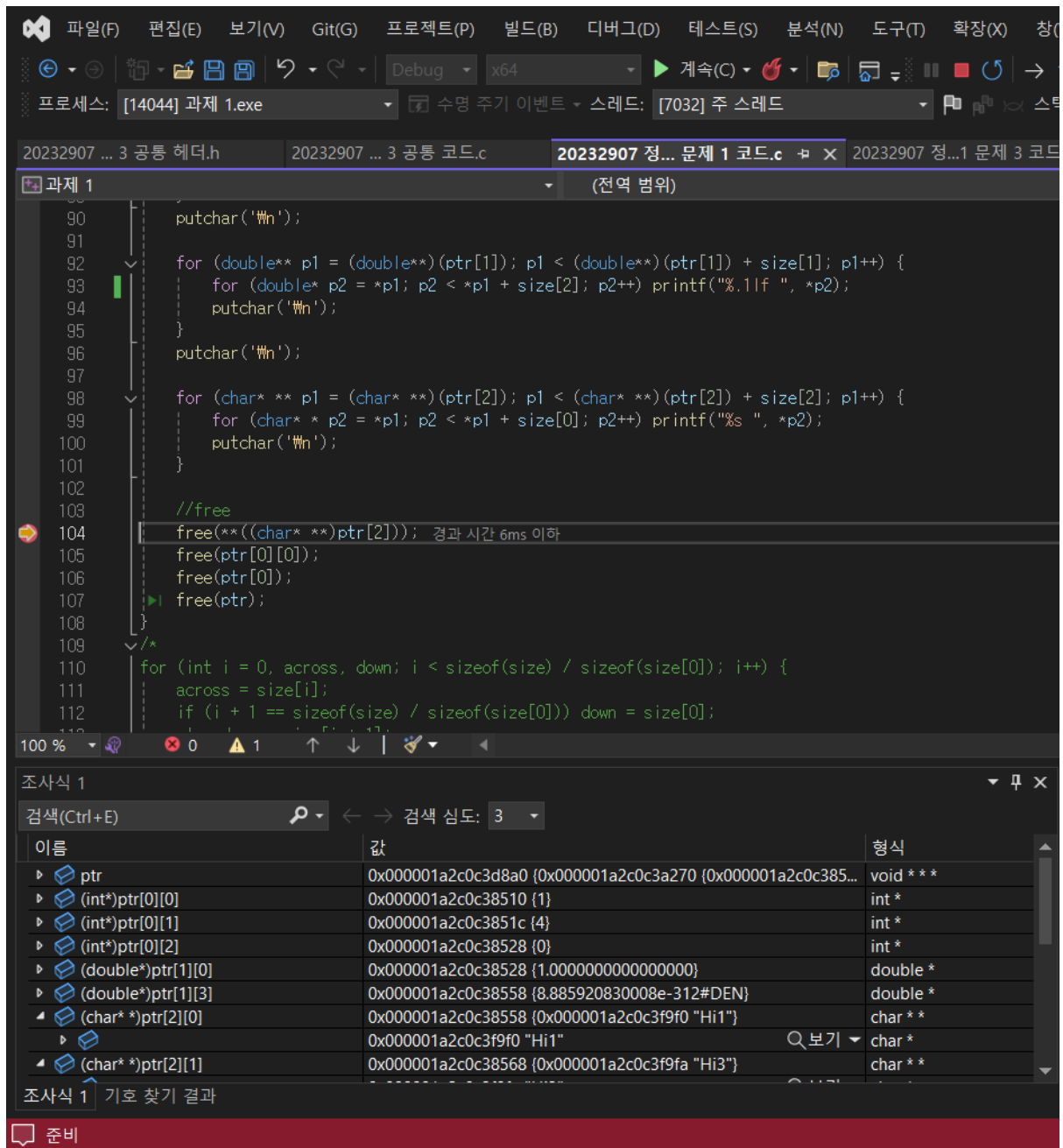


그림 3.5

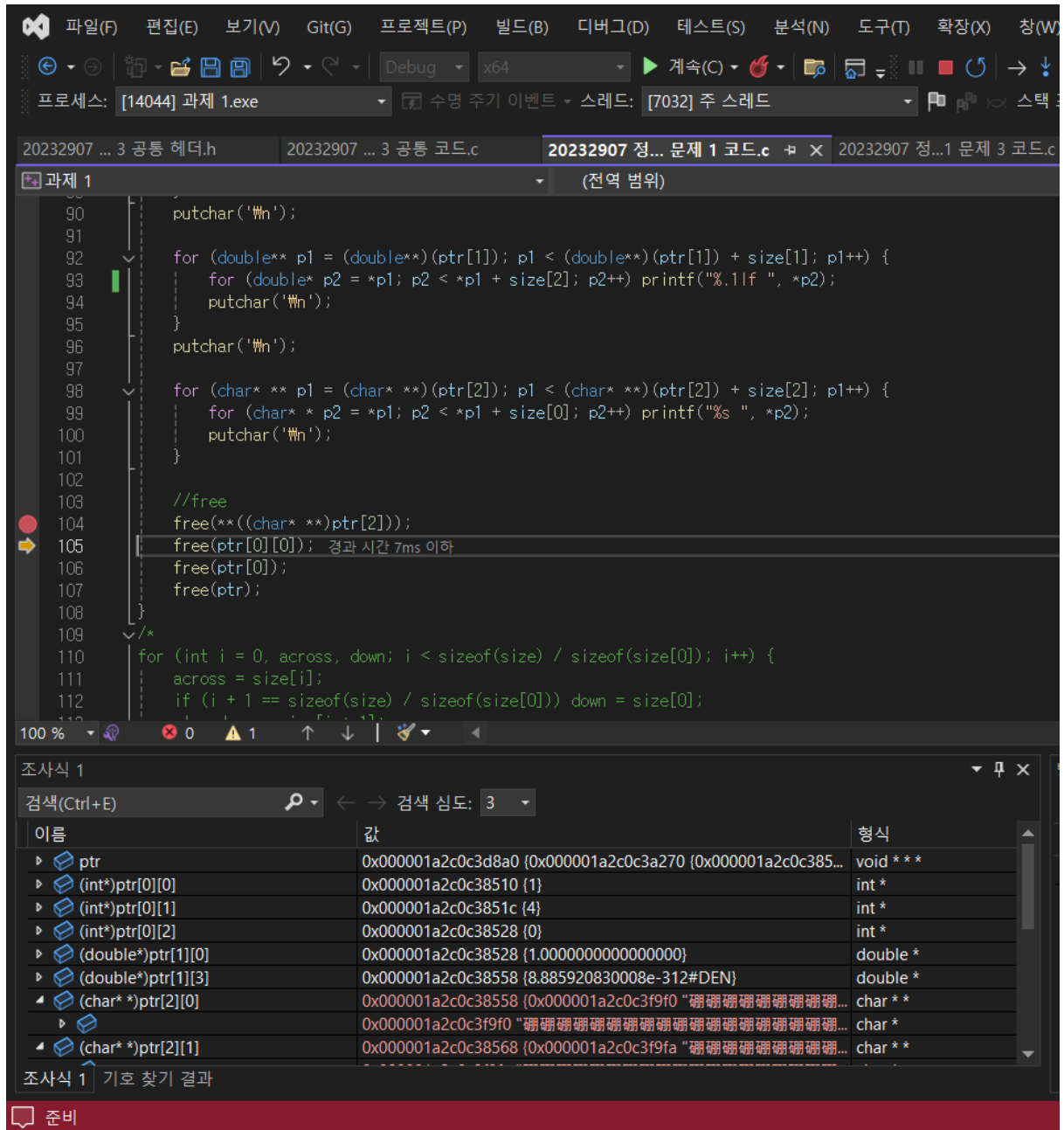


그림 3.6

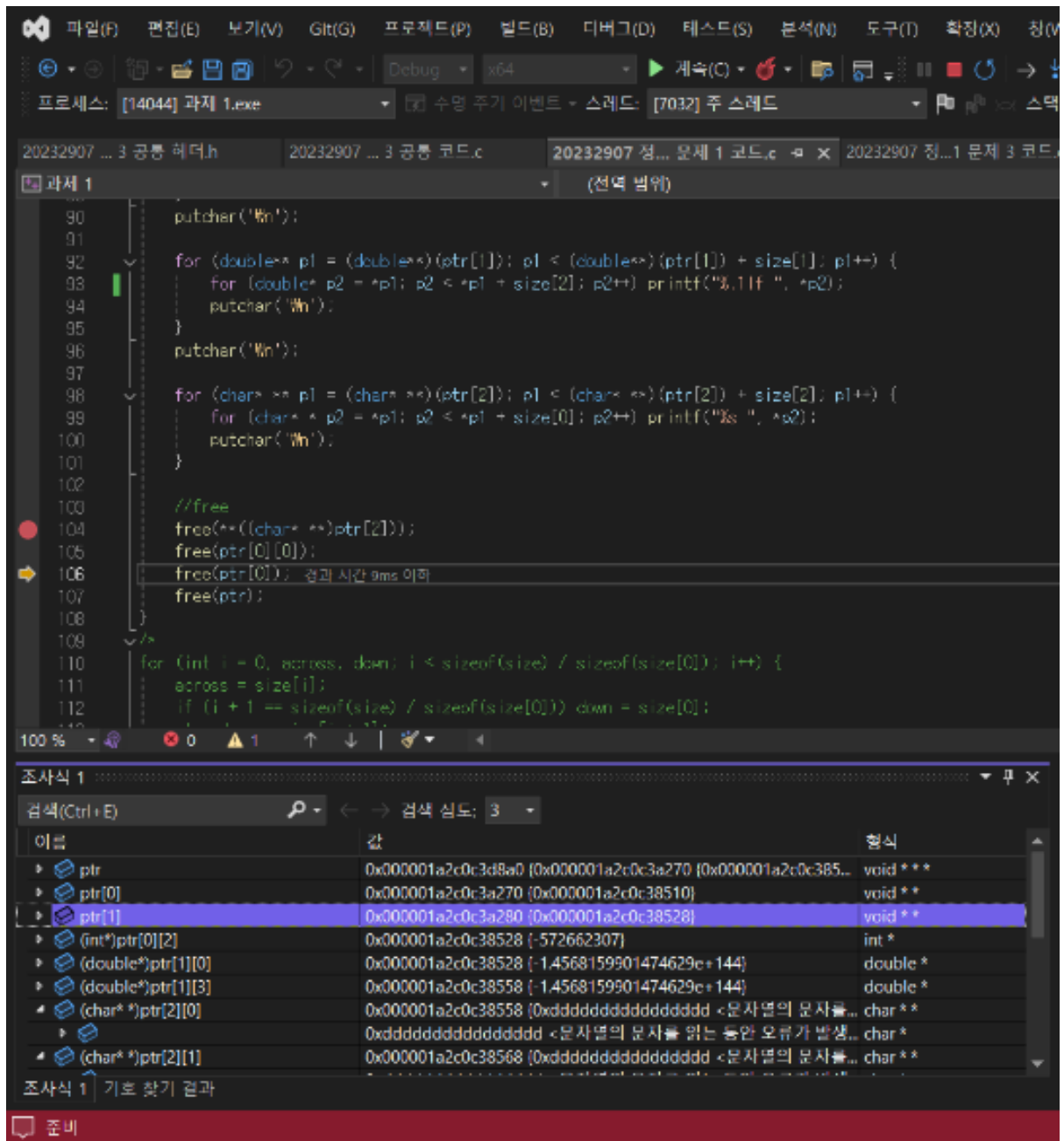


그림 3.7

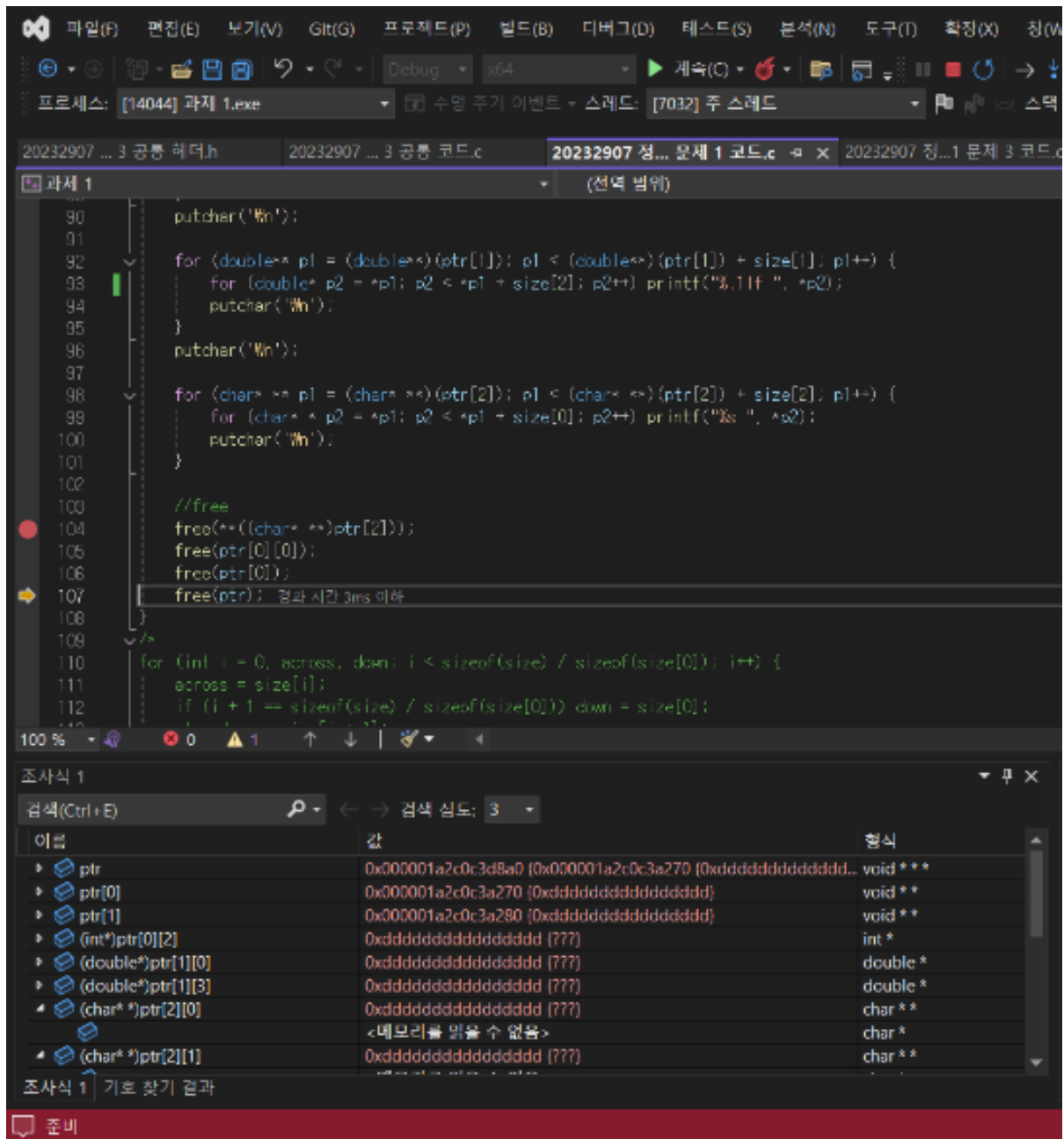


그림 3.8

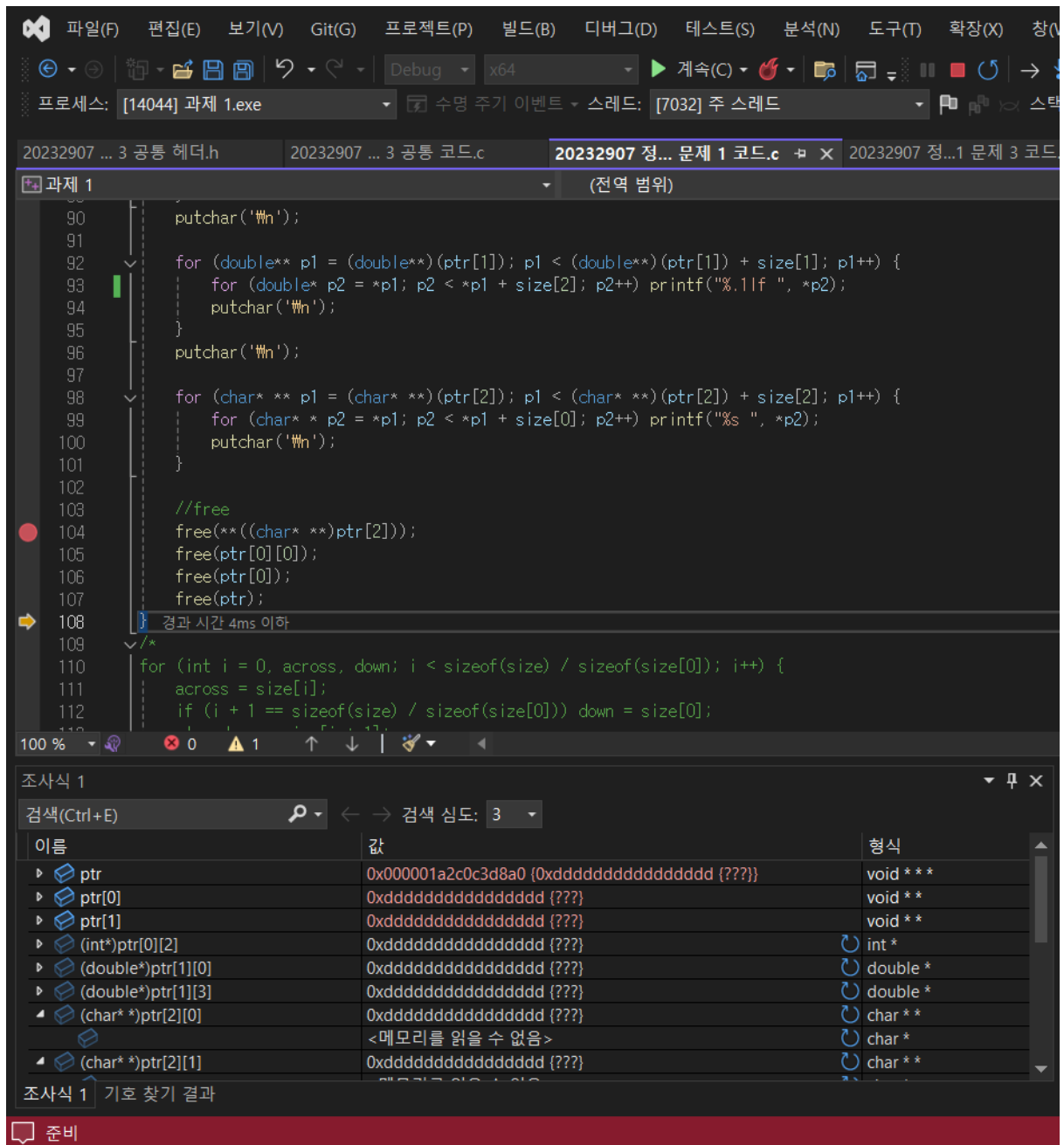


그림 3.9

할당 해제도 정상적으로 처리되어 오류가 발생하지 않았고, 해제한 공간도 해제 즉시 이상한 값으로 채워진다는 점을 보면, 해제가 정상적으로 이루어졌다는 점을 알 수 있다. (해제 즉시 이상한 값으로 채워지는 게 정상이라고 단정할 수 없지만, 적어도 해제가 이루어지지 않았다면 이상한 값으로 채워질 리가 없기 때문에 이를 근거로 한 것이다.)

이외에도 숫자를 바꾸어서 몇 번 더 실행해 보았다.

```

Microsoft Visual Studio 디버그
a, b, c? 1 1 1
1

1.0

Hi1

```

그림 3.10

```

Microsoft Visual Studio 디버그
a, b, c? 9 9 9
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81

1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7
2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6
3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5
4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4
5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.2 6.3
6.4 6.5 6.6 6.7 6.8 6.9 7.0 7.1 7.2
7.3 7.4 7.5 7.6 7.7 7.8 7.9 8.0 8.1
8.2 8.3 8.4 8.5 8.6 8.7 8.8 8.9 9.0

Hi1 Hi2 Hi3 Hi4 Hi5 Hi6 Hi7 Hi8 Hi9
Hi10 Hi11 Hi12 Hi13 Hi14 Hi15 Hi16 Hi17 Hi18
Hi19 Hi20 Hi21 Hi22 Hi23 Hi24 Hi25 Hi26 Hi27
Hi28 Hi29 Hi30 Hi31 Hi32 Hi33 Hi34 Hi35 Hi36
Hi37 Hi38 Hi39 Hi40 Hi41 Hi42 Hi43 Hi44 Hi45
Hi46 Hi47 Hi48 Hi49 Hi50 Hi51 Hi52 Hi53 Hi54
Hi55 Hi56 Hi57 Hi58 Hi59 Hi60 Hi61 Hi62 Hi63
Hi64 Hi65 Hi66 Hi67 Hi68 Hi69 Hi70 Hi71 Hi72
Hi73 Hi74 Hi75 Hi76 Hi77 Hi78 Hi79 Hi80 Hi81

```

그림 3.11

```

Microsoft Visual Studio 디버그
a, b, c? 2 3 1
1 2 3
4 5 6

1.0
1.1
1.2

Hi1 Hi2

```

그림 3.12

```

Microsoft Visual Studio 디버그
a, b, c? 3 4 3
1 2 3 4
5 6 7 8
9 10 11 12

1.0 1.1 1.2
1.3 1.4 1.5
1.6 1.7 1.8
1.9 2.0 2.1

Hi1 Hi2 Hi3
Hi4 Hi5 Hi6
Hi7 Hi8 Hi9

```

그림 3.13

```

Microsoft Visual Studio 디버그
a, b, c? 9 1 9
1
2
3
4
5
6
7
8
9

1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8

Hi1 Hi2 Hi3 Hi4 Hi5 Hi6 Hi7 Hi8 Hi9
Hi10 Hi11 Hi12 Hi13 Hi14 Hi15 Hi16 Hi17 Hi18
Hi19 Hi20 Hi21 Hi22 Hi23 Hi24 Hi25 Hi26 Hi27
Hi28 Hi29 Hi30 Hi31 Hi32 Hi33 Hi34 Hi35 Hi36
Hi37 Hi38 Hi39 Hi40 Hi41 Hi42 Hi43 Hi44 Hi45
Hi46 Hi47 Hi48 Hi49 Hi50 Hi51 Hi52 Hi53 Hi54
Hi55 Hi56 Hi57 Hi58 Hi59 Hi60 Hi61 Hi62 Hi63
Hi64 Hi65 Hi66 Hi67 Hi68 Hi69 Hi70 Hi71 Hi72
Hi73 Hi74 Hi75 Hi76 Hi77 Hi78 Hi79 Hi80 Hi81

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\x64\Debug\과제 1.exe(프로세스 13040개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...|

```

그림 3.14

```

Microsoft Visual Studio 디버그
a, b, c? 8 5 2
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40

1.0 1.1
1.2 1.3
1.4 1.5
1.6 1.7
1.8 1.9

Hi1 Hi2 Hi3 Hi4 Hi5 Hi6 Hi7 Hi8
Hi9 Hi10 Hi11 Hi12 Hi13 Hi14 Hi15 Hi16

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\x64\Debug\과제 1.exe(프로세스 14988개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...|

```

그림 3.15

위와 같이 모든 경우에서 정상 동작하는 것을 알 수 있다.

## 4. 문제 2: 파이썬의 List 내부 구조 파악

파이썬 List의 내부 구조를 파악한다. 파이썬은 C/C++로 작성되었으므로, 파이썬의 List와 동일한 구조를 C언어에서 구현할 수 있다. 이를 위해 파이썬의 내부 구조를 알아보고 정리하여 장점을 설명한다. 별도의 코딩은 필요 없다.



## 5. 문제 2 파이썬 List의 내부 구조

파이썬의 모든 자료형은 PyObject 구조체의 확장이며, 여기에 각 자료형의 크기가 저장되어 있다. 정수를 python에서 int라 쓰고 부르지만, 그 밑에는 int로 선언된 게 아니다. 모든 자료형을 PyObject로 처리하는 것에서 다양한 자료형이 들어갈 수 있는 이유가 설명된다. List 원소의 개수만 별도로 저장하며, 여러 가지 원소는 PyObject의 확장판이고 실제 데이터는 또 동적 할당의 형태로 저장하므로, 원소별로 데이터 크기는 걱정할 필요가 없다. 모두 포인터인 만큼 같은 크기를 가진다. 또 list의 공간은 동적 할당받으며, 처음에는 할당하지 않다가 공간이 부족할 때 4, 8, 16, 24, 32, 40, 52, ... 바이트의 순서로 공간을 늘린다. 원소의 삭제로 필요한 공간에 비해 너무 많은 공간을 할당받았으면, 할당받은 공간의 1/2 이하로 필요한 시점에서 크기를 조정한다. 이렇게 구현하는 경우 너무 많은 공간을 할당받아 남은 공간도 별로 없거나, 너무 적은 공간을 할당받아 데이터의 확장을 막는 문제 둘 다 막을 수 있어 데이터도 절약하고, 저장하는 데이터의 크기도 자유자재로 변경할 수 있다. 아래 문제 3이나, 저번 학기 프로그래밍 과제 3, 5에서 문자열을 받을 때 구현하고 사용했던 sgets 함수나, 바로 아래 문제 3에서 다항식의 저장 방식을 개선할 때와 비슷하다(이에 대한 설명은 문제 3 해결 방법에서도 다시 설명한다). 이 방식의 linkedlist 대비 장점은 특정 원소를 찾기 쉽다는 점이 있다. Linkedlist에 5개의 원소가 차례대로 저장되어 있다고 하면, 포인터로 찾아가는 것을 5회 해야 문제를 해결할 수 있다. 반면 배열로 5개의 원소가 저장되어 있다면, 마지막 원소를 찾는 것은 배열의 시작점 포인터 위치에 +4만 해 주면 된다. 특히 stack으로 사용되는 python의 list 특성상 마지막 원소를 찾을 일이 많을 수 있기에 이는 매우 큰 장점으로 볼 수 있다. 이외에도 JAVA의 Garbage Collector처럼 기존 list 전체를 버리면 알아서 공간을 해제해 주는 시스템도 있어 list 전체를 바꾸는 것도 별도의 메모리 해제 작업 없이 쓸 수 있다는, 코드를 작성하는 입장에서 편하다는 장점도 있다. 단 list 중간에 값을 넣거나 특정 원소를 삭제하는 것은, linkedlist 방식이라면 포인터 두 개만 조정해 주면 되지만, 배열로 저장하는 방식이라 하나하나 값을 옮겨야 하는 단점은 있을 것으로 보인다.

## 6. 문제 3: 다항식의 저장 방식 개선

다항식의 0이 아닌 항만을 저장할 때, 미리 충분한 개수의 항을 설정하여 배열로 그 공간을 받는 것은 비효율적이다. 이를 개선하고 다항식의 덧셈과 곱셈을 구현한다.

## 7. 문제 3 해결 방안

배열로 공간을 받는 문제는 동적 할당으로 해결했다. 처음에는 작은 크기의 공간만 할당받고, 필요할 때마다 늘리는 방식이다. 다만 조금씩 늘리는 게 아니라 약 3개의 항이 들어갈 정도의 공간을 한 번에 할당받는다. 한 개의 항이 더 추가될 때 한 개의 항이 들어갈 정도의 공간만 할당

받으면 항이 추가될 때마다 할당을 새로 받아야 해 시간이 오래 걸릴 것이다. 그렇다고 한 번에 너무 많은 공간을 할당받으면, 충분한 개수의 항을 설정하여 배열로 그 공간을 받는 문제상황과 다를 바가 없어진다. 따라서 공간 문제는 동적 할당으로, 한 번에 적당한 크기의 공간을 받고 필요하면 더 늘려서 쓰는 방식으로 해결했다.

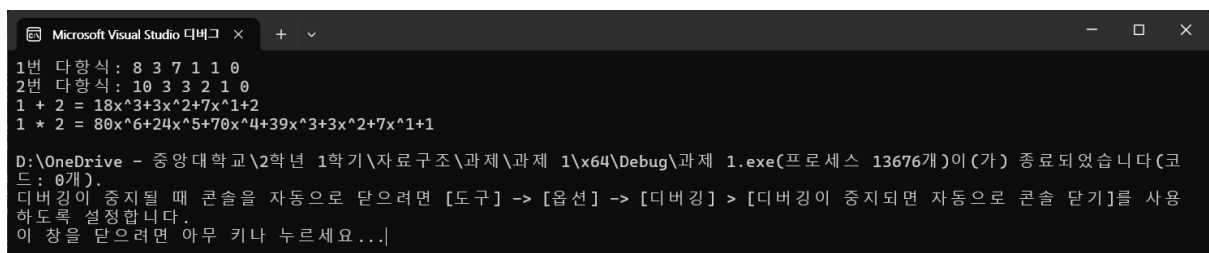
강의자료에서 다항식의 계수가 실수형이고 지수가 정수형이므로 이 둘을 저장하는 구조체를 선언하고, 이 구조체 배열의 포인터와 동적 할당받은 길이, 현재 사용하고 있는 길이를 저장하는 구조체도 선언했다. 동적 할당을 다루는 건 다항식의 항을 저장하는 함수를 따로 만들어 거기서 했는데, 할당이 되어 있지 않다면 할당하고, 크기가 부족하다면 재할당하며, 할당받은 크기가 변할 때마다 구조체에 같이 저장된, 동적 할당된 크기 값을 변경한다. 물론 할당 작업 이후에는 다항식의 항을 저장한다. 프로그램을 실행하면 입력을 문자열로 받은 후, 문자열을 기반으로 두 개의 식을 생성한다. 이때 문자열 입력은 줄 변경 문자가 들어올 때까지 계속 입력받는다. 문자열을 입력받는 공간도 동적 할당으로 받고 fgets 함수를 사용해 할당받은 공간까지만 입력받았으며, 다 입력받은 게 아니라면 재할당을 통해 줄 변경 문자가 들어올 때까지 값을 계속 입력받았다. 이 문자열 입력 방식은 저번 학기 프로그래밍 과목의 과제 3, 과제 5와 같으며, 헤더파일 이름 수정과 freeset 함수의 오류를 고친 것 외에는 모든 것이 동일하다. 이후 띄어쓰기 단위로 끊어 각각 지수와 차수로 저장했다. 저장하고 다항식을 동적 할당받는 과정은 앞에서 언급한 함수가 모두 처리하므로, 각종 문제가 되는 상황은 고려하지 않아도 된다. 다만 지수만 있고 차수가 입력되지 않았을 때 차수에 0을 넣어주는 처리와, 다항식의 항이 아예 없는 경우 에러 코드를 넣어주는 것, 그리고 동적 할당에 실패해 함수가 NULL을 반환할 때만 짧은 처리를 해 주면 된다. 다항식의 항이 아예 없는 경우 에러 코드를 넣어주는 것은 그냥 놔둘 경우 구조체에 저장된 포인터 값이 NULL이 되어, 동적 할당에 실패했을 때와 구별할 수 없다는 문제 때문이다. 이때의 에러 코드는 NULL도 아니고 동적 할당받은 공간의 주소도 되지 않을 값으로 적당히 골라서 넣어주면 되며, 여기서는 0xFF를 사용했다. 다항식의 항에 접근할 때는 할당된 길이 또는 저장된 포인터 값을 먼저 확인해, 0xFF 주소의 데이터를 읽는 과정을 피하여 프로그램이 죽는 일이 없도록 하였다. (즉 포인터가 가리키는 값만 보고 그 위치의 데이터를 읽지 않는 방법으로 에러를 막았다.) 이외에도 입력이 차수 기준으로 정렬되어 들어온다는 것을 가정하고 있으나, 그렇지 않을 때를 대비해 다항식을 정리해 주었다. 즉 문제의 요구사항은 아니나 사태를 대비해 코드를 추가하였다. 아래 곱셈에서 쓴 함수를 재활용하는 것이니만큼 코드가 그리 복잡해지지도 않는다. 다항식을 정리하는 과정은 곱셈을 설명할 때 자세히 설명한다.

다음 이 두 다항식의 덧셈과 곱셈을 진행하였다. 덧셈은 다항식이 차수 기준으로 정렬되어 있다는 점을 이용하여 두 다항식을 앞에서부터 하나씩 보고 차수가 큰 순서대로 채워 넣었으며, 차수가 같은 경우 두 항의 지수를 더해 저장하였다. 단 지수의 덧셈 결과가 0이면 그 차수는 저장하지 않았다. 곱셈은 한번 곱할 때 정렬을 하기가 어려운 관계로, 일단 순서 상관없이 모두 곱했다. 그러니까 항이 3개인 다항식 2개끼리 곱하면 정리하기 전의 항 9개를 모두 저장한다는 말이

다. 이후 곱셈을 검토해 보며, 같은 차수인 항이 여러 개 있으면 하나의 항으로 묶고, 이후 항을 정렬하는 과정을 거쳐 곱셈을 완료했다. 곱셈을 이렇게 한 이유는, 곱셈 결과로 나올 수 있는 최대 차수 항부터 1씩 차수를 낮추어 가며 곱셈 결과 중 해당 차수가 있는지 확인하고 결과를 계산하는 방법도 있지만, 이는 메모리 측면에서만 효율적이지 시간 소요는 더 많이 늘어난다고 보아 시행하지 않았으며(특히 최대 차수가 매우 높을 때 더욱 그렇다), 이외 다른 알고리즘은 딱히 마땅한 것이 떠오르지도 않고 그것을 적용하는 게 더 복잡해질 것 같아 이렇게 하였다. 다항식의 곱셈 정의가 한 식의 한 항에서 다른 식의 모든 항을 곱하는 과정을 반복한다는 점으로 생각해 보면 이 방법은 곱셈의 정석이기도 하다. 이후의 복잡한 다항식에서도 이 정의를 따라 곱셈한다는 점도 고려하여 다음과 같이 코드를 작성하였다. 참고로 다항식의 항의 개수는 할당 길이보다 작아야 하므로, 덧셈과 곱셈 과정에서 0이 아닌 항이 아예 없는 다항식에 대한 처리는 별도로 해 주지 않아도 된다. 다항식의 항의 개수가 0임을 확인하면 다항식의 항에 전혀 접근하지 않기 때문이다.

마지막으로 결과를 출력하고, 할당받은 공간을 모두 해제한 뒤 프로그램을 종료하였다.

## 8. 문제 3의 결과

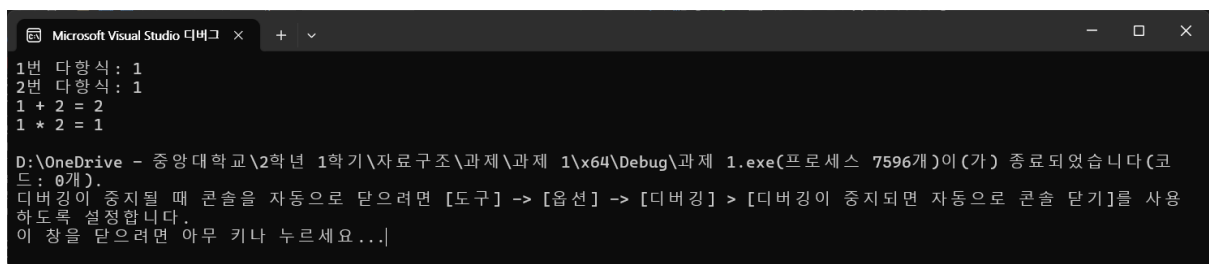


```

Microsoft Visual Studio 디버그
1번 다항식: 8 3 7 1 1 0
2번 다항식: 10 3 3 2 1 0
1 + 2 = 18x^3+3x^2+7x^1+2
1 * 2 = 80x^6+24x^5+70x^4+39x^3+3x^2+7x^1+1
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\Debug\과제 1.exe(프로세스 13676개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
  
```

그림 8.1

위 그림 8.1은 요구사항의 입력 예시를 그대로 입력한 것이다. 출력 결과가 요구사항과 같다는 점을 알 수 있다.



```

Microsoft Visual Studio 디버그
1번 다항식: 1
2번 다항식: 1
1 + 2 = 2
1 * 2 = 1
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\Debug\과제 1.exe(프로세스 7596개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
  
```

그림 8.2

차수가 들어오지 않을 때의 예외 처리를 진행했으므로 위 그림 8.2처럼 정수의 계산도 문제가 없다.

```

Microsoft Visual Studio 디버그
1번 다항식: 9 200000000 -5 -100000000
2번 다항식: 7 100000000 -3 -200000000 2
1 + 2 = 9x^200000000+7x^100000000+2-5x^-100000000-3x^-200000000
1 * 2 = 63x^300000000+18x^200000000-62-10x^-100000000+15x^-300000000

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 19232개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.3

위 그림 8.3은 차수가 클 때의 결과를 나타낸 것이다. 차수가 커져도 계산에 문제가 없다는 점을 볼 수 있다.

```

Microsoft Visual Studio 디버그
1번 다항식: 7.2 64 5.6 22 6.1 7
2번 다항식: -7.2 64 5.6 22 -8.8 2
1 + 2 = 11x^22+6x^7-9x^2
1 * 2 = -52x^128-44x^71-63x^66+31x^44+34x^29-49x^24-54x^9

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 16428개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.4

위 그림 8.4와 같이 합이나 곱이 0이 되는 경우도 정상적으로 해당 항을 결과 표출 시 제외한다는 점도 볼 수 있다. 다만 출력 시 소수점 미만은 반올림 처리하였기에 소수점이 제대로 나오지 않는 건 문제 되는 상황이 아니다.

```

Microsoft Visual Studio 디버그
1번 다항식:
2번 다항식:
1 + 2 = 0
1 * 2 = 0

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 22024개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.5

```

Microsoft Visual Studio 디버그
1번 다항식: 1 3
2번 다항식:
1 + 2 = 1x^3
1 * 2 = 0

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 10860개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.6

그림 8.5, 그림 8.6은 하나 또는 둘 다 다항식이 0일 때의 결과를 나타낸 것이다. 입력이 안 들어왔을 때 0을 처리하는 것부터 계산하는 것까지 정상적으로 동작함을 알 수 있다.

이 이외에 2번 더 계산해 보았다.

```

Microsoft Visual Studio 디버그
1번 다항식: 1 1 1 0
2번 다항식: 1 1 1 0
1 + 2 = 2x^1+2
1 * 2 = 1x^2+2x^1+1
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 19528개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.7

```

Microsoft Visual Studio 디버그
1번 다항식: 9 9 8 8 7 7 6 6 5 5 4 4 3 3 2 2 1 1
2번 다항식: 1.1 9 2.2 8 3.3 7 4.4 6 5.5 5 6.6 4 7.7 3 8.8 2 9.9 1 11
1 + 2 = 10x^9+10x^8+10x^7+10x^6+10x^5+11x^4+11x^3+11x^2+11x^1+11
1 * 2 = 10x^18+29x^17+55x^16+88x^15+126x^14+169x^13+216x^12+264x^11+314x^10+363x^9+304x^8+246x^7+192x^6+143x^5+99x^4+62x^3+32x^2+11x^1
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 21488개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...

```

그림 8.8

그림 8.8의 곱셈 결과 확인을 위해 wolframalpha 계산기를 돌려 본 결과는 다음과 같다. 소수점 이하 반올림을 고려하면 결과가 일치함을 확인할 수 있다. (이 계산 결과도 소수점 1째자리까지 반올림되어 있어, 똑같이 .5로 끝나는데 어떤 건 올리고 어떤 건 내리는 것은 문제 되지 않는다.)

Expanded form

$$9.9x^{18} + 28.6x^{17} + 55x^{16} + 88x^{15} + 126.5x^{14} + 169.4x^{13} + 215.6x^{12} + 264x^{11} + 313.5x^{10} + 363x^9 + 303.6x^8 + 246.4x^7 + 192.5x^6 + 143x^5 + 99x^4 + 61.6x^3 + 31.9x^2 + 11x$$

Step-by-step solution

그림 8.9

위와 같이 모든 경우에서 정상 동작한다는 점을 알 수 있다.

## 9. 문제 4: 희소 행렬 저장 방식의 곱셈

희소행렬 저장 방식을 이용해 해당 방식으로 저장된 자료형의 곱셈을 구현한다.

## 10. 문제 4 해결 방안

우선 입력 방식은, 처음에 행렬의 크기(가로, 세로 크기)를 입력받고, 그 후 0이 아닌 항의 세로 위치, 가로 위치, 값을 순서대로 입력받는 방식으로 하였다. 값이 0인 항이 들어오면 입력을 종료하였다. 데이터를 저장하는 방식은 위 문제 3처럼 하나의 항 정보를 담는 구조체(row, column, value를 멤버변수로 가지는 SingleMatrix 구조체) 배열을 동적 할당받아 저장했으며, 동적 할당을

한 번 할 때마다 8개의 항을 저장할 수 있는 공간씩을 할당받았다. 할당받은 크기와 사용하는 크기, 행렬의 가로, 세로 크기도 같이 묶어 구조체 SparseMatrix에 저장하였다. 이 방식으로 행렬 2개를 입력받은 후, 곱셈을 진행하였다. 먼저 행렬의 곱셈 정의에 따라 행렬의 곱셈이 가능한지 확인하고, 곱셈이 불가능한 경우 곱셈 과정을 진행하지 않고, 에러 코드를 담은 행렬 구조체를 반환했다. 에러 코드는 위 문제 3과 같은 방식으로, 행렬의 값을 동적 할당받아 저장한 위치를 나타내는 포인터에 특정 값을 넣어주면서 전달했다.

곱셈은 (행렬 1)\*(행렬 2), (행렬 2)\*(행렬 1) 모두 진행하였으며, 곱셈 과정은 글로만 작성해서는 이해하기 매우 어려울 것으로 예상되니 그림과 예시를 같이 첨부해 설명한다. 예시로 쓸 두 행렬은 다음과 같다.

앞쪽 배열(6x6)			뒤쪽 배열(6x6)		
행 (column)	열 (row)	값 (value)	행 (column)	열 (row)	값 (value)
0	3	7	0	3	12
1	0	9	1	3	6
1	5	8	1	5	4
3	0	6	3	0	2
3	1	5	3	1	10
4	5	1	4	5	16
5	2	2	5	2	8

그림 10.1

뒤쪽에 붙는 행렬은 곱셈 계산 전에 전치를 진행하였다. 전치를 진행한 이유는 곱셈 시 정렬을 진행하지 않기 위함이며, 위 문제 3에서 다항식의 곱셈을 끝낸 후 식을 정리했던 것을 곱셈 전 정리한 것으로 생각하면 된다. (코드 작성 시에는 전치를 했지만, 단순히 정렬만 바뀌어도 문제가 없다. 정렬 변경용 qsort를 위한 matrix\_sortbyrow도 만들어 놓았으나 사용하지 않았다.) 그림 10.2는 예시 행렬의 전치 결과이다.

행 (column)	열 (row)	값 (value)	뒤쪽 배열	행 (column)	열 (row)	값 (value)
행 (column)	열 (row)	값 (value)		행 (column)	열 (row)	값 (value)
0	3	12	→	0	3	2
1	3	6		1	3	10
1	5	4		2	5	8
3	0	2		3	0	12
3	1	10		3	1	6
4	5	16		5	1	4
5	2	8		5	4	16

그림 10.2

이후 먼저 앞쪽에 붙는 행렬에서 같은 행(세로좌표, 즉 column 값이 같은 곳)인 항의 범위를 찾아 행렬에서 시작점과 끝점을 저장했다. 시작점과 끝점 저장은 포인터를 이용했고, 각 포인터

변수의 이름은 시작점 po1s, 끝점 po1d로 하였다. 이 po1s와 po1d는 후술하는 po1이 가리킬 수 있는 항의 범위가 된다. (행이 동일한 항만을 보기 위한 조치이다.) 이렇게 po1s와 po1d를 이용해 항의 범위를 제한하지 않으면 결과값의 순서가 중구난방이 될 수 있고, 동일한 좌표의 항이 2개 이상 저장되는 문제가 일어날 수 있다. 아래 그림 10.3은 그 결과이다. 이 예시에서는 행이 0인 항이 하나밖에 없어 시작점과 끝점 사이에 들어가는 항이 단 한 개이지만, 아래 그림 10.9같이 행이 1인 항이 여러 개라면 시작점과 끝점 사이에 들어가는 항이 여러 개가 될 수 있다.

행 (column)	열 (row)	값 (value)
0	3	7
1	0	9
1	5	8
3	0	6
3	1	5
4	5	1
5	2	2

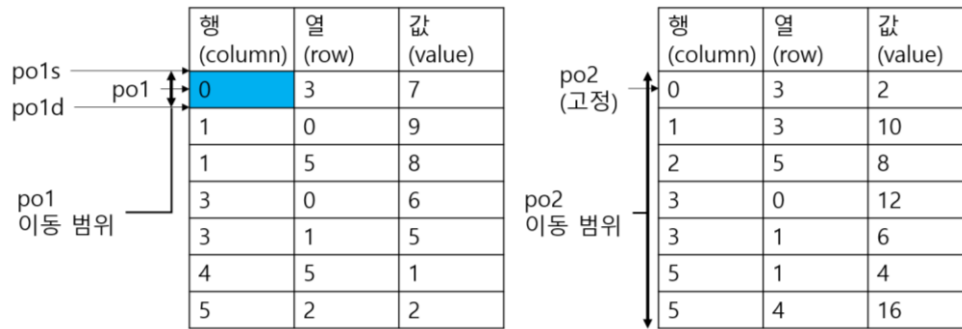
그림 10.3

다음으로 전치된 뒤쪽 행렬의 항 위치를 하나하나 살펴 보면서 앞에서 저장된 앞 행렬의 시작점과 끝점 중 열(가로좌표, 즉 row 값)이 같은 항이 있는지 살핀다. (원래 앞쪽 항의 열과 뒤쪽 행렬의 행 위치가 같아야 하지만, 전치를 진행하였기에 둘 다 행 위치가 같은지 보면 된다.) 아래 그림 10.4, 그림 10.5는 그 과정을 나타낸 것으로, 우선 뒤 행렬의 항을 가리키는 포인터 po2를 고정한 후, 앞 행렬의 항을 가리키는 포인터 po1을 움직인다. (그러니까 이중 for문에서, 바깥 for문에서 po2를 뒤 행렬의 첫 항부터 마지막 항까지 차례대로 움직이고, 안쪽 for문에서 po1을 po1s에서 po1d 직전까지 움직인다.) 예시에서는 아직 po1이 움직일 수 있는 범위는 한 행밖에 없어 po1이 움직이지 않는 것으로 보일 수 있다.

행 (column)	열 (row)	값 (value)
0	3	7
1	0	9
1	5	8
3	0	6
3	1	5
4	5	1
5	2	2

행 (column)	열 (row)	값 (value)
0	3	2
1	3	10
2	5	8
3	0	12
3	1	6
5	1	4
5	4	16

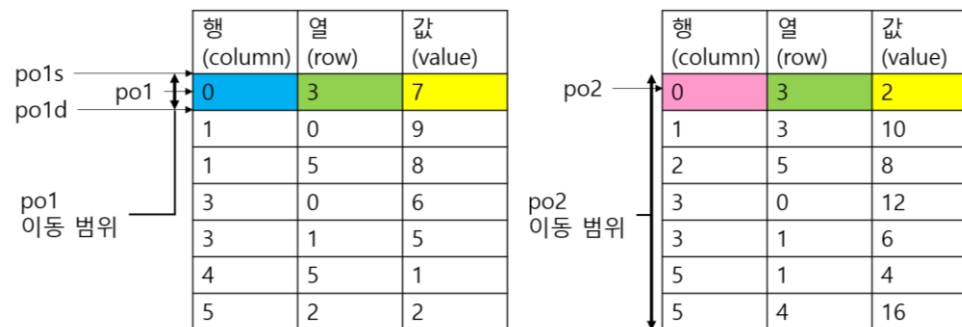
그림 10.4



(현재 po1s=po1,  
한 항목에 훑을 수 없음)

그림 10.5

해당하는 항목이 있다면 계산을 진행한다. 앞 행렬의 행 좌표가 곱셈값의 행 좌표가 되고, 뒤 행렬의 열 좌표가 곱셈값의 열 좌표가 된다. (뒤 행렬의 열 좌표가 곱셈값의 열 좌표가 되어야 하지만 앞에서 언급했듯 전치를 진행하였다.) 곱셈값의 좌표를 저장한 후 그 값을 저장한다. 값은 두 항목 값의 곱이다. 아래 그림 10.6은 그 결과이다. tmpres는 곱셈값 항목 하나를 저장하는 구조체 변수이다.



(현재 po1s=po1,  
한 항목에 훑을 수 없음)

tmpres	
행(column)	0
열(row)	0
값(value)	14

그림 10.6

이때 이렇게 저장된 곱셈값(행, 열, 값으로 구성된 구조체 하나)은 바로 결과를 저장하는 행렬에 넣지 않는데, 이 이유는 같은 좌표에 해당하는 결과가 더 있을 수도 있기 때문이며, 이는 예시에서 해당하는 사례가 나왔을 때 자세히 설명한다. 이 작업을 뒤 행렬의 모든 항목을 훑을 때까지 반복한다. 열 값이 같은 항목이 더 나오면 곱셈값을 저장하는데 곱셈값의 좌표가 이전과 같다면(즉 이전에 계산한 결과와 대비해 두 행렬의 열 위치만 다르고 행 위치는 같은 경우) 좌표값 저장 없이 두 항목 곱한 결과를 기존 값에 더하고, 좌표가 다르다면 이미 기존 좌표의 곱셈값 계산이 끝났으므로 기존 결과를 최종 결과를 저장하는 행렬에 저장한 뒤, 좌표와 값을 새로 저장한다. 그림 10.7~그림 10.8은 그 과정을 나타낸 것이다.



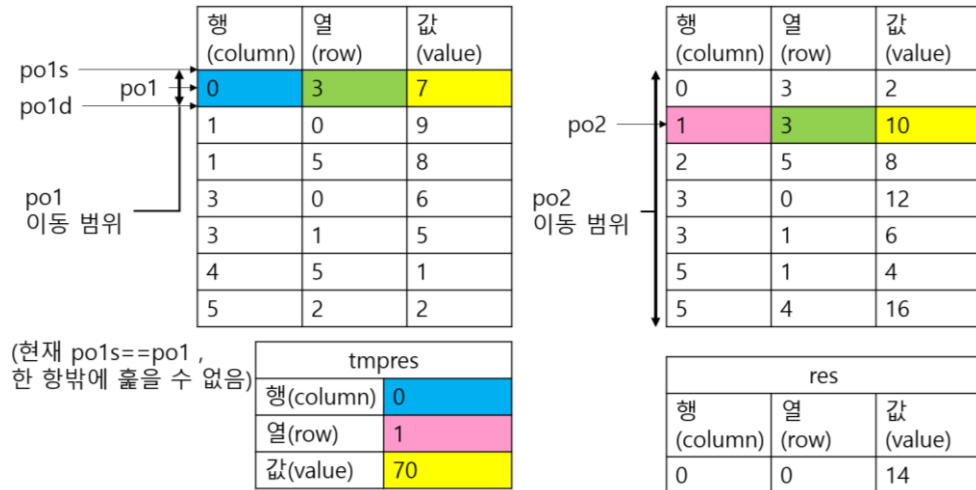


그림 10.7

열이 같은 두 항을 찾았으나 기존 tmpres에 저장된 것과 좌표가 달라, 기존 tmpres에 저장된 값은 결과 행렬을 저장하는 res에 넣고, 새롭게 찾은 두 항을 곱한 결과를 저장한다.

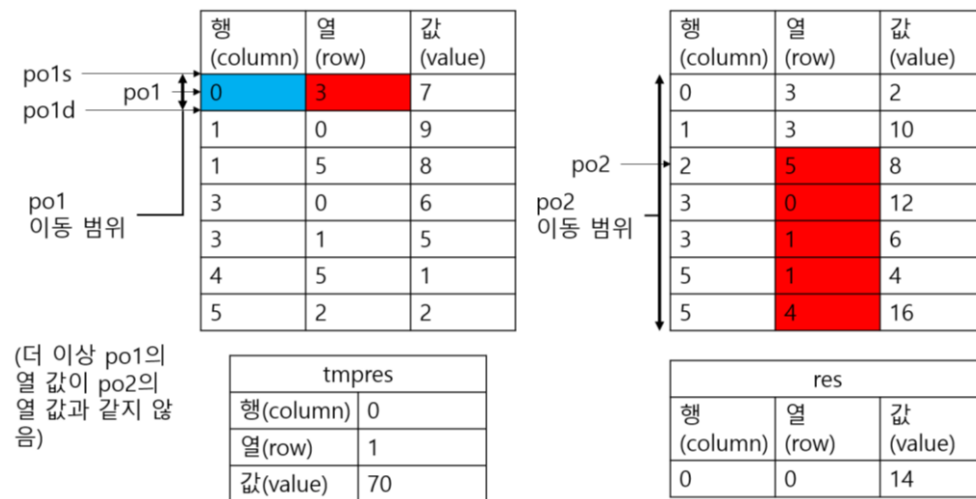


그림 10.8

po1s, po1d가 움직이기 전까지는, 이후 뒤 행렬과 po1이 가리킬 수 있는 항 중에서 열이 일치하는 항(빨간색 음영으로 표시된 부분 참고)이 없다. 이렇게 뒤 행렬의 모든 항을 훑었으면 앞 행렬의 항을 가리키는 포인터 두 개를 움직여 다음 행을 가리키도록 하고, 위 작업을 반복한다. 아래 그림 10.9~그림 10.19는 그 과정이다.

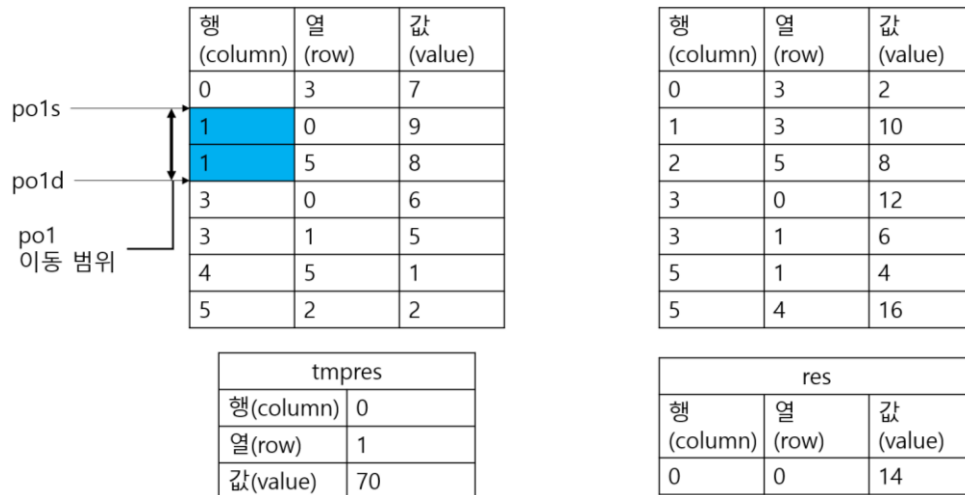


그림 10.9

po1s, po1d를 움직였다. 이제 po1가 가리킬 수 있는 항은 행 값이 1인 항 2개가 되었다. 앞서 말했듯 po2의 값을 하나 정한 후 po1가 가리킬 수 있는 항을 하나씩 보며 열이 일치하는 게 있는지 확인한다.



그림 10.10

po2가 그림 10.10의 위치에 오기 전까지는 po1이 가리킬 수 있는 항과 열이 일치하는 게 없어 연산이 불가능하므로 해당 과정은 생략한다. po2가 그림 10.10의 위치에 고정되어 있고 po1을 움직일 때, po1이 가리키는 항이 그림 10.10과 같다면, 열 값이 다르므로 연산할 수 없다. (빨간색 음영으로 표시된 부분 참고) 그러나 po1이 움직여 그림 10.11과 같은 항을 가리킨다면, po1과 po2가 가리키는 두 항의 열 값이 같으므로 연산을 진행할 수 있다. 앞서서처럼 기존에 tmpres에 저장된 좌표와 현재 좌표가 다르므로, 기존 tmpres에 저장된 결과는 res로 옮기고 새로운 좌표의 연산 결과를 저장한다.

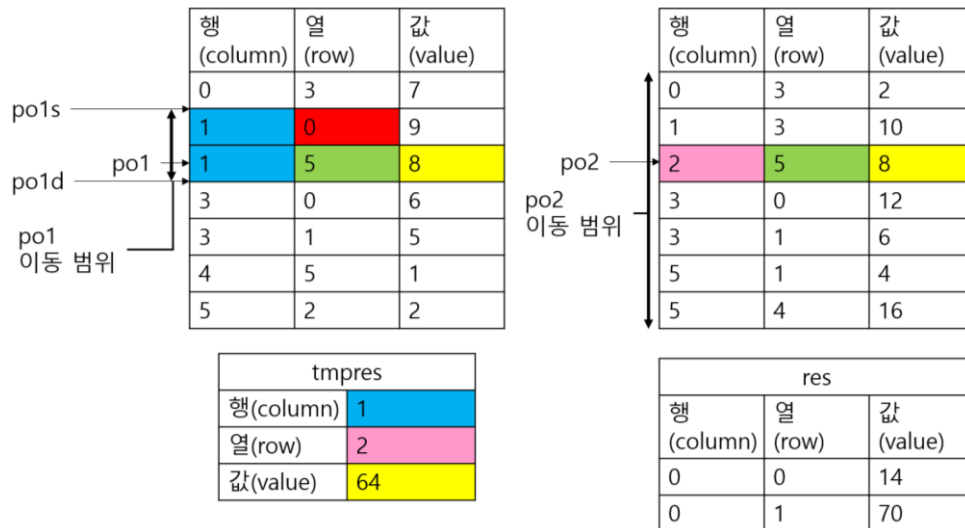


그림 10.11

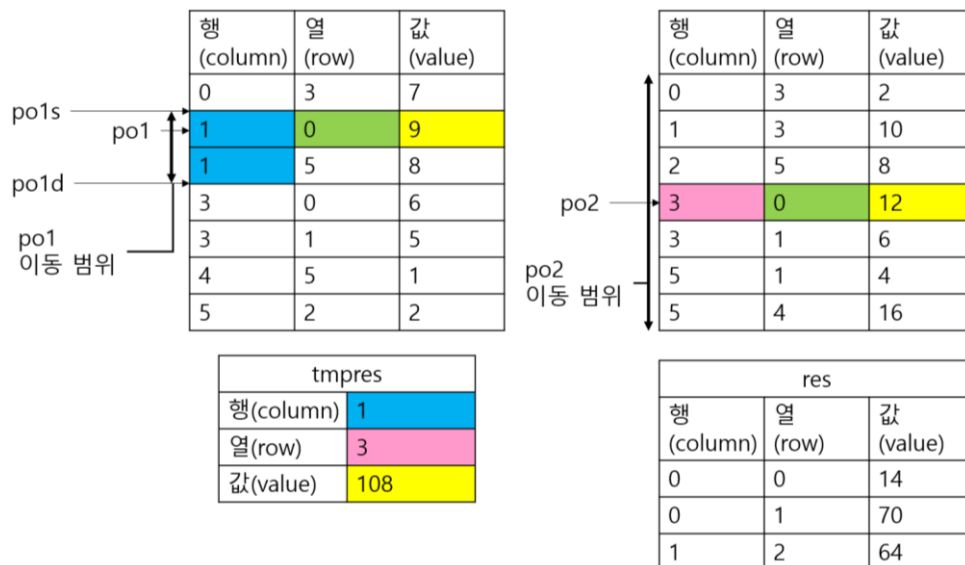


그림 10.12

이후 po2가 움직여 위 그림 10.12와 같은 상황이 되었다. 역시 좌표가 기존과 다르므로 기존 값은 res로 옮기고 새로 연산한 값을 저장한다.

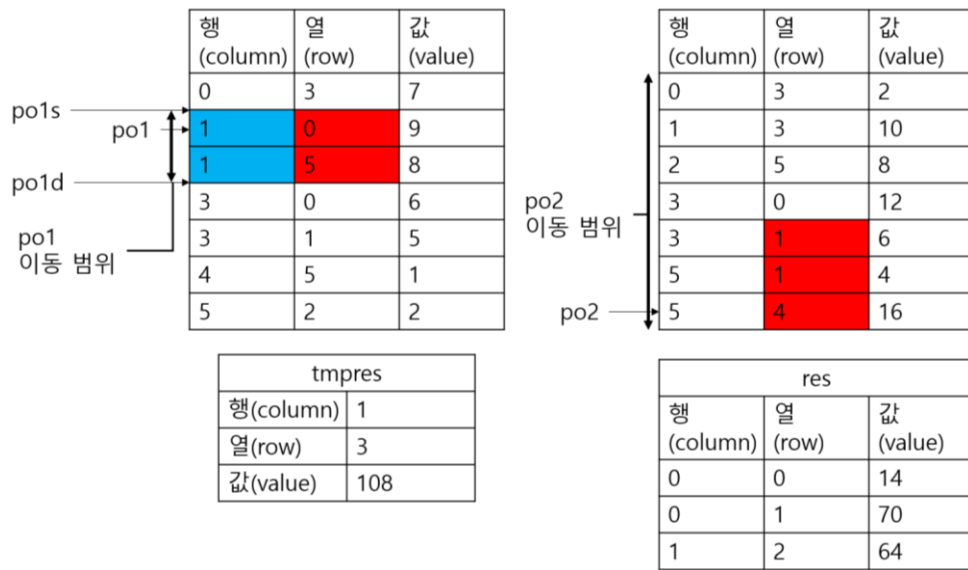


그림 10.13

이후 그림 10.8처럼 더 이상 일치하는 열이 없기 때문에 그림 10.14처럼 po1s, po1d가 가리키는 위치를 이동한다.

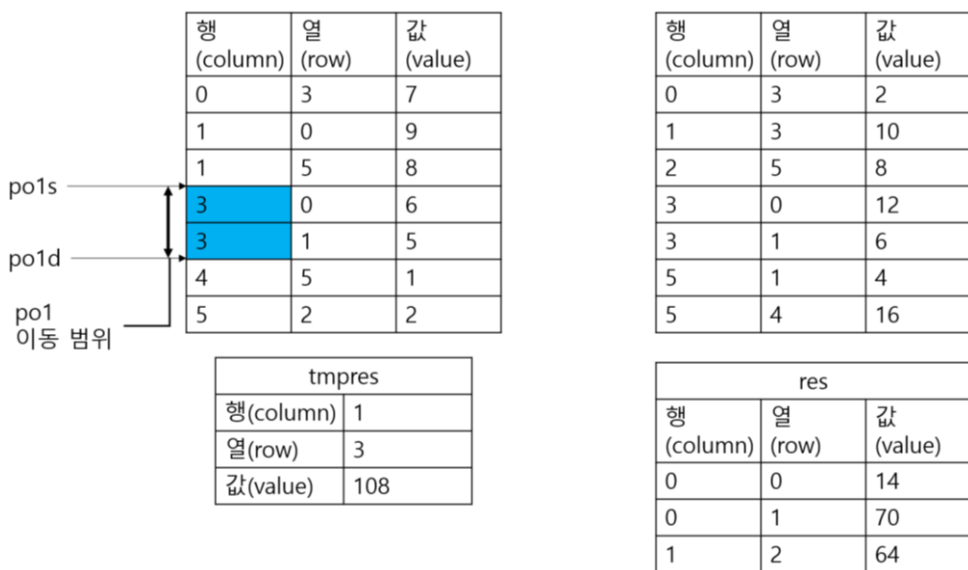


그림 10.14

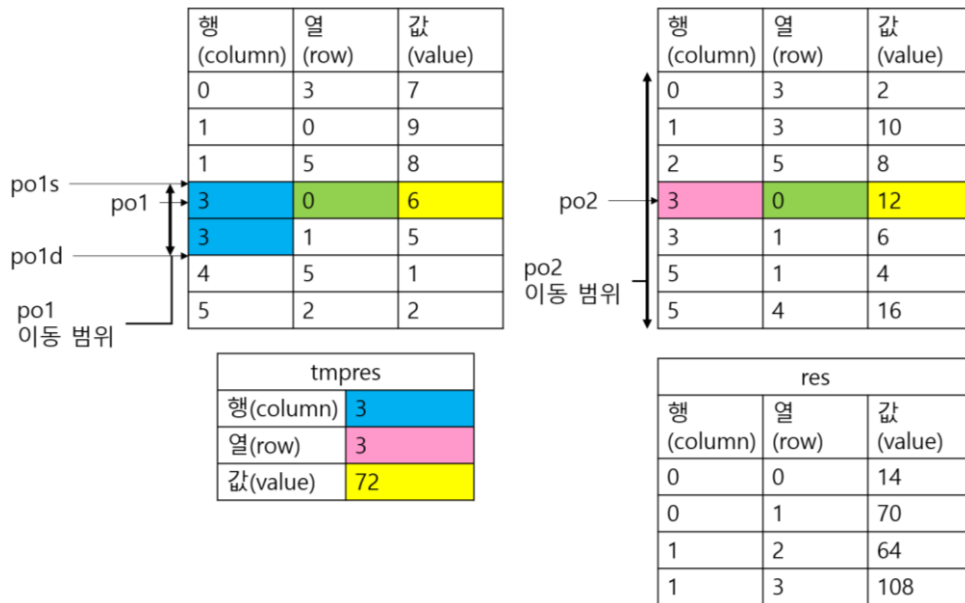


그림 10.15

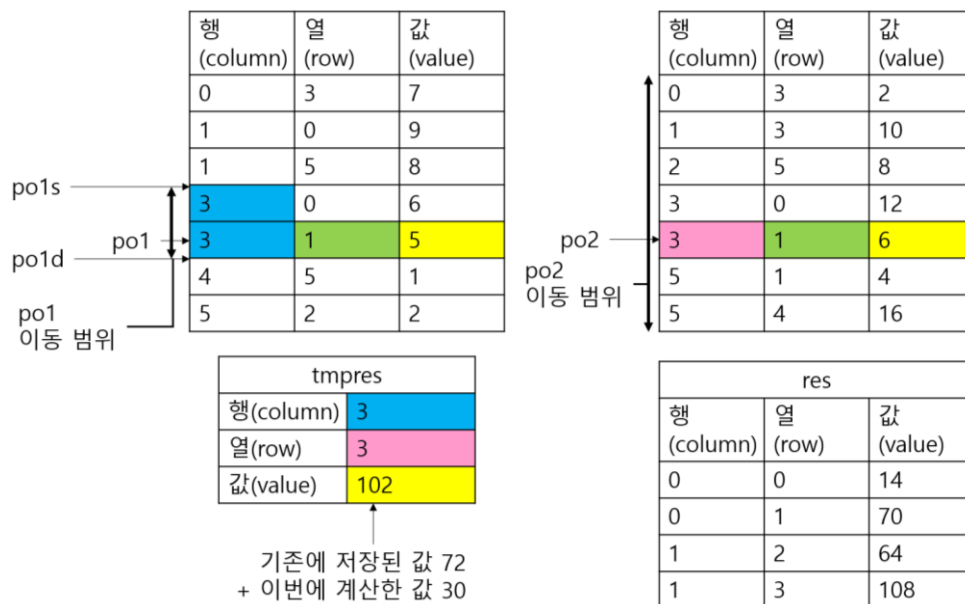


그림 10.16

위 그림 10.15는 평범하게 tmpres에 값을 저장한다. 그런데 그림 10.16처럼 po2와 po1이 이동하면, 이번에 연산한 결과가 들어갈 좌표와 이미 tmpres에 저장된 항의 좌표가 같다. 이 경우는 저장된 결과값과 이번에 계산한 결과값을 더해준다. 이는 행렬의 정의에 의한 것이다.

$[a_1 \cdots a_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = a_1x_1 + \cdots + a_nx_n$ 인데, 그림 10.16 이전까지는  $a_1x_1, \cdots, a_nx_n$  중 0이 아닌 항이 없거나 1개밖에 없었다. 그런데 그림 10.15, 그림 10.16에서는 0이 아닌 항이 2개 있는 것이다. 이때는 두 곱셈 결과값을 더해줘야 한다. 이러한 이유로 tmpres에 저장된 연산의 결과를 바로 res에 넣지 않는 것이다. 0이 아닌 항이 여러 개 있을 경우 0이 아닌 항을 모두 더한 뒤에 그 값

을 res에 넣기 위함이다. 앞에서 언급했던 "결괏값의 좌표가 이전과 같다면(즉 이전에 계산한 결과와 대비해 두 행렬의 열 위치만 다르고 행 위치는 같은 경우) 좌표값 저장 없이 두 항을 곱한 결과를 기존 값에 더한다"는 말이 바로 이것을 말하는 것이다.

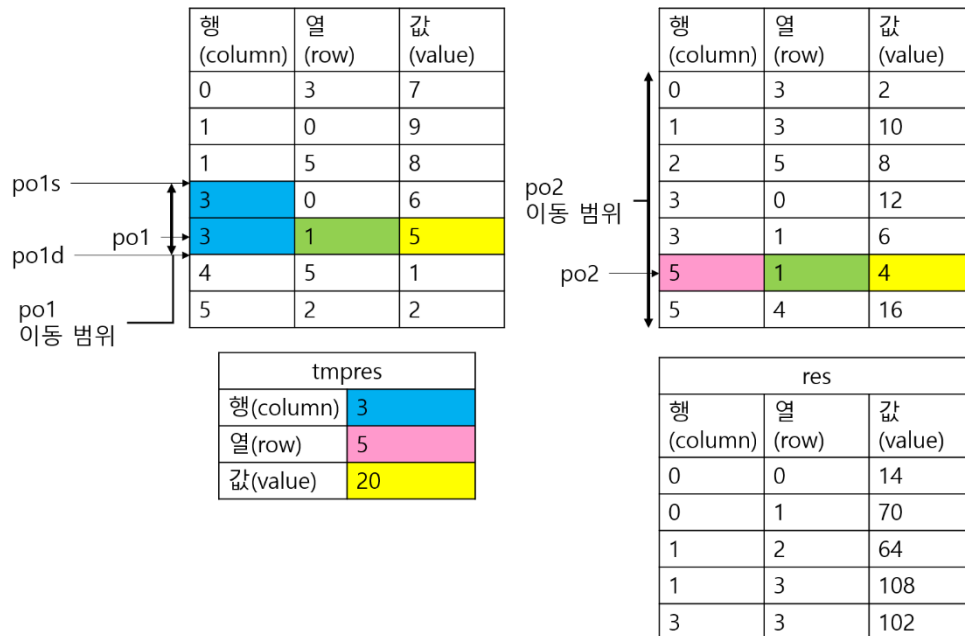


그림 10.17

이제 결괏값의 좌표가 달라졌으므로 기존 값을 res에 저장하고 새로운 좌표의 연산 결과를 저장한다. 이후 같은 방법으로 계속 진행한다.

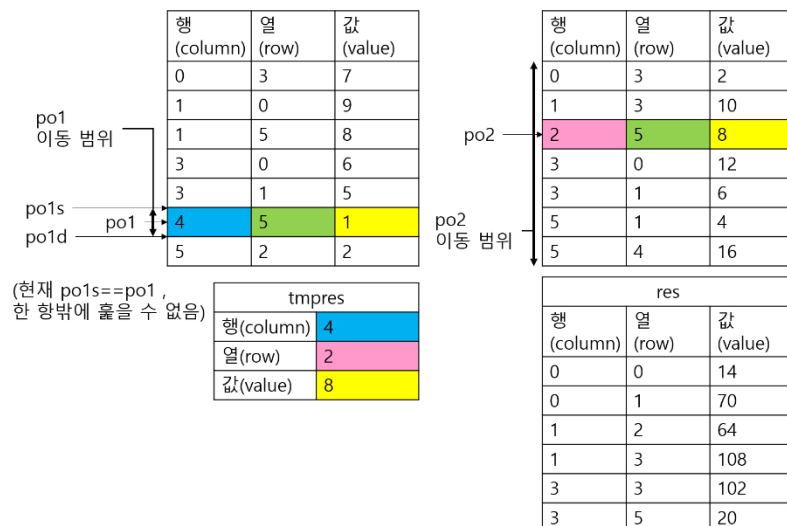


그림 10.18

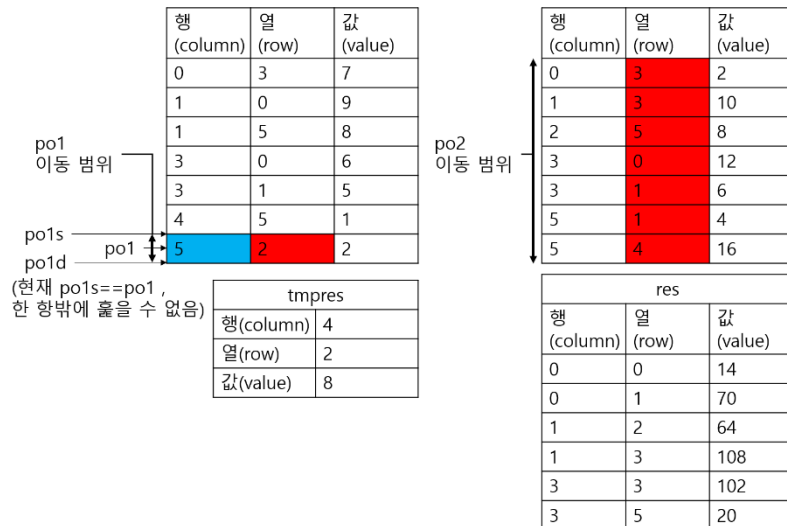


그림 10.19

po1d가 가리키는 항이 앞 행렬의 0이 아닌 마지막 항을 가리키는데(정확히는 마지막 항 직후의 항이다. po1과 비교 연산 시  $\leq$ 이 아닌  $<$  연산을 사용하기 때문이다.) po2와 po1의 항 중 열이 일치하는 항을 찾지 못했다. 이렇게 앞 행렬의 모든 항도 훑어 시작 지점을 가리키는 포인터가 앞 행렬에 저장된 항을 넘겼다면 곱셈을 완료한 것이다. 그림 10.20처럼 마지막에 남아있던 곱셈값을 최종 결과를 저장하는 행렬에 넣어주고 그 결과를 반환한다.

res		
행 (column)	열 (row)	값 (value)
0	0	14
0	1	70
1	2	64
1	3	108
3	3	102
3	5	20
4	2	8

tmpres		
행 (column)	열 (row)	값 (value)
4	2	8

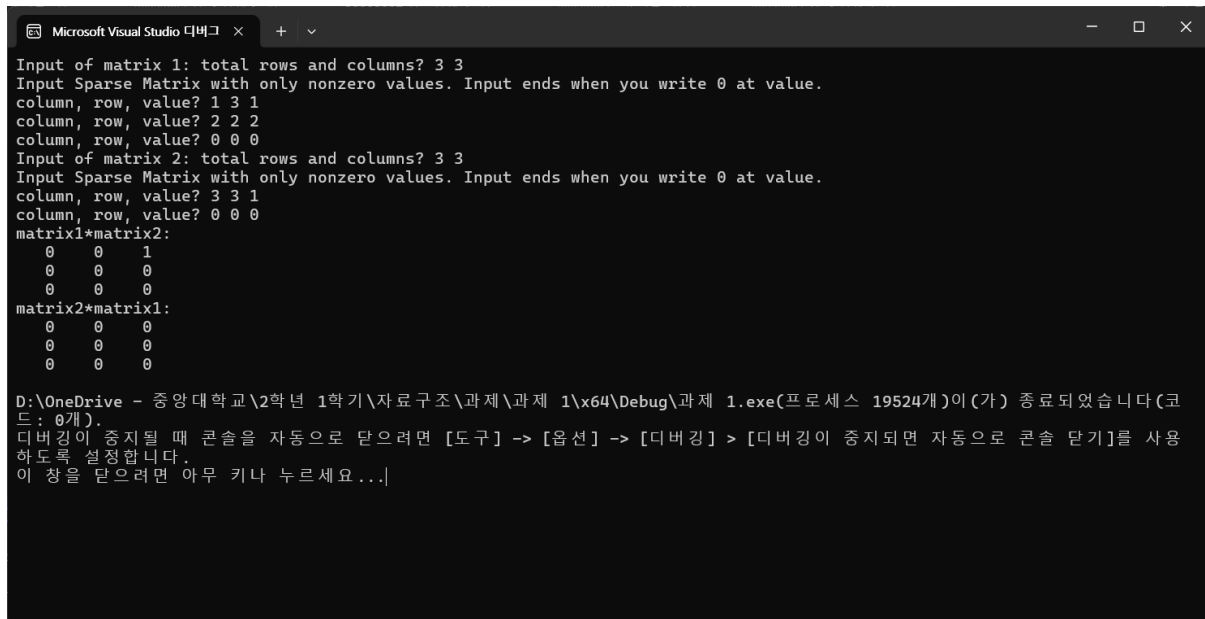
그림 10.20

마지막에 남아있던 곱셈값이 없다면 tmpres 변수가 한 번도 값이 변하지 않은, 즉 곱셈의 결과는 모든 항이 0인 행렬이 되므로 에러 코드를 담아 그 결과를 반환한다. 모든 항이 0인 행렬에 대해 별도의 처리가 있는 이유는 에러 코드를 담지 않으면 동적 할당에 실패한 경우인지 항이 없는 건지 구별할 방법이 없기 때문이다.

이렇게 (행렬 1)\*(행렬 2), (행렬 2)\*(행렬 1) 곱셈을 모두 완료한 후, 각 계산의 결과를 출력했다. 결과의 출력은 일반적인 행렬을 출력하듯이 값이 0인 항을 포함해 모든 항을 출력하도록 했다. (입력 양식처럼 출력할 때를 대비해 그 방식으로 출력하는 코드도 만들어 두었지만 사용하지 않고 주석처리했다.) 존재할 수 없는 행렬인 경우 행렬 출력 대신 행렬이 존재할 수 없다는 메시지만 출력했다. 이후 동적 할당받은 모든 공간을 해제하고 프로그램을 종료하였다.

위와 같은 방법으로 문제 4를 해결했다.

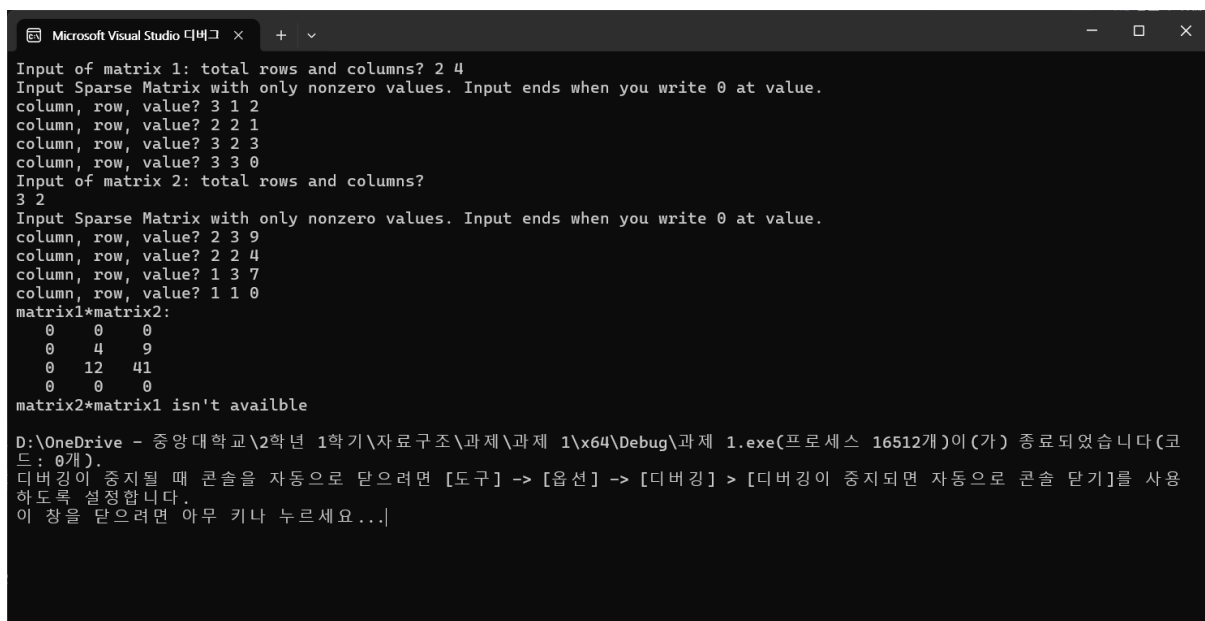
## 11. 문제 4의 결과



```
Microsoft Visual Studio 디버그
Input of matrix 1: total rows and columns? 3 3
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 1 3 1
column, row, value? 2 2 2
column, row, value? 0 0 0
Input of matrix 2: total rows and columns? 3 3
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 3 3 1
column, row, value? 0 0 0
matrix1*matrix2:
0 0 1
0 0 0
0 0 0
matrix2*matrix1:
0 0 0
0 0 0
0 0 0
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\x64\Debug\과제 1.exe(프로세스 19524개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요 ...|
```

그림 11.1

간단한 행렬 계산 결과이다.



```
Microsoft Visual Studio 디버그
Input of matrix 1: total rows and columns? 2 4
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 3 1 2
column, row, value? 2 2 1
column, row, value? 3 2 3
column, row, value? 3 3 0
Input of matrix 2: total rows and columns?
3 2
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 2 3 9
column, row, value? 2 2 4
column, row, value? 1 3 7
column, row, value? 1 1 0
matrix1*matrix2:
0 0 0
0 4 9
0 12 41
0 0 0
matrix2*matrix1 isn't available
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\x64\Debug\과제 1.exe(프로세스 16512개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요 ...|
```

그림 11.2



```
Microsoft Visual Studio 디버그 × + ~
Input of matrix 1: total rows and columns? 3 2
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 2 3 9
column, row, value? 2 2 4
column, row, value? 1 3 7
column, row, value? 1 1 0
Input of matrix 2: total rows and columns? 2 4
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 3 1 2
column, row, value? 2 2 1
column, row, value? 3 2 3
column, row, value? 3 3 0
matrix1*matrix2 isn't available
matrix2*matrix1:
0 0 0
0 4 9
0 12 41
0 0 0

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\Debug\과제 1.exe(프로세스 18328개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...|
```

그림 11.3

그림 11.3은 그림 11.2에서 행렬의 순서만 바꾼 것이다. 둘 다 계산 결과가 올바르다는 점을 확인할 수 있다.

```
Microsoft Visual Studio 디버그 × + ~
Input of matrix 1: total rows and columns? 6 6
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 1 4 7
column, row, value? 2 1 9
column, row, value? 2 6 8
column, row, value? 4 1 6
column, row, value? 4 2 5
column, row, value? 5 6 1
column, row, value? 6 3 2
column, row, value? 7 7 0
Input of matrix 2: total rows and columns? 6 6
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 1 4 12
column, row, value? 2 4 6
column, row, value? 2 6 4
column, row, value? 4 1 2
column, row, value? 4 2 10
column, row, value? 5 6 16
column, row, value? 6 3 8
column, row, value? 7 7 0
matrix1*matrix2:
14 70 0 0 0 0
0 0 64 108 0 0
0 0 0 0 0 0
0 0 0 102 0 20
0 0 8 0 0 0
0 0 0 0 0 0
matrix2*matrix1:
72 60 0 0 0 0
36 30 8 0 0 0
0 0 0 0 0 0
90 0 0 14 0 80
0 0 32 0 0 0
0 0 0 0 0 0

D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\Debug\과제 1.exe(프로세스 18600개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록
설정합니다.
이 창을 닫으려면 아무 키나 누르세요...|
```

그림 11.4

```

Microsoft Visual Studio 디버그
Input of matrix 1: total rows and columns? 6 6
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 1 4 12
column, row, value? 2 4 6
column, row, value? 2 6 4
column, row, value? 4 1 2
column, row, value? 4 2 10
column, row, value? 5 6 16
column, row, value? 6 3 8
column, row, value? 7 7 0
Input of matrix 2: total rows and columns? 6 6
Input Sparse Matrix with only nonzero values. Input ends when you write 0 at value.
column, row, value? 1 4 7
column, row, value? 2 1 9
column, row, value? 2 6 8
column, row, value? 4 1 6
column, row, value? 4 2 5
column, row, value? 5 6 1
column, row, value? 6 3 2
column, row, value? 7 7 0
matrix1*matrix2:
72 60 0 0 0 0
36 30 8 0 0 0
0 0 0 0 0 0
90 0 0 14 0 80
0 0 32 0 0 0
0 0 0 0 0 0
matrix2*matrix1:
14 70 0 0 0 0
0 0 64 108 0 0
0 0 0 0 0 0
0 0 0 102 0 20
0 0 8 0 0 0
0 0 0 0 0 0
D:\OneDrive - 중앙대학교\2학년 1학기\자료구조\과제\과제 1\64\Debug\과제 1.exe(프로세스 18740개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.

```

그림 11.5

그림 11.4는 앞 문제 4 해결 방안에서 예시로 들었던 배열이고, 그림 11.5는 행렬의 순서만 바꾼 것이다. 둘 다 계산 결과가 올바르다는 점을 확인할 수 있다.

## 12. 문제 5: 미로 저장하기

미로를 최소한의 크기로 저장할 방법을 제안한다. 설명만 진행하고 별도의 코딩은 필요 없다. 적어도 200바이트, 가능하면 50바이트 이내의 크기로 미로를 저장한다.

## 13. 문제 5 해결 방안

움직일 수 있는 공간을 기준으로 칸을 그리면 주어진 미로는  $9 \times 12 = 108$ 칸이다. 이 칸에서 벽이 존재할 수 있는 곳은 각 칸의 상하좌우로 총 4곳이다. 따라서 이 칸을 기준으로 상하좌우에 벽이 있는지 저장하려고 한다. 그런데 여기서 인접한 칸은 벽을 공유한다. 즉 한 칸과 그 칸의 오른쪽 칸은 사이에 벽이 될 수 있는 공간이 하나 있고, 그 사이에 벽이 있냐 없냐는 정보는 양쪽 칸에 저장된 값이 같을 수밖에 없다. 왼쪽 칸에는 칸 오른쪽에 벽이 있다고 저장되어 있지만 오른쪽 칸에는 칸 왼쪽에 벽이 없다고 저장된 건 말이 안 된다. 여기서 이 하나의 공간(벽이 될 수 있는 공간)에 벽이 있는지를 굳이 2개의 칸에 나누어 저장할 필요는 없다. 여기에 미로 외부는 뚫린 공간 없이 모두 막혀 있으므로, 미로 테두리의 벽 정보를 저장할 필요는 없다. **따라서 칸 하나당 칸 오른쪽과 아래쪽에 벽이 있는지만 저장하면 된다.** 칸 왼쪽에 벽이 있는지는 왼쪽 칸의 정보를, 칸 위쪽에 벽이 있는지는 위쪽 칸의 정보를 확인하면 된다. 1열과 1행의 왼쪽, 위쪽 칸 정보는, 미로

외부로 뚫려있는 벽이 없다는 점을 이용해, 파일에 해당 정보가 없더라도 모두 벽이 있는 것으로 처리할 수 있다. 여기에 마지막 행과 열은 각각 아래쪽과 오른쪽에 벽이 있는지를 저장할 필요가 없으므로 용량을 더 줄일 수 있다. 이렇게 되면 미로의 크기(행과 열)를 저장하기 위해 각 1바이트(unsigned char, 0~255까지의 정수 저장 가능)를 사용한다고 해도, 1칸당 2비트를 사용하므로 주어진 미로 파일은  $(2*9*12-1*9-1*12) \div 8 + 2 = 26.375 \approx 27$ 바이트면 미로를 저장할 수 있다. 저장 방식을 그림으로 대략 나타내면 다음과 같다. 맨 앞 2비트에 미로의 크기를 저장하고, 그 이후 맨 오른쪽 칸과 맨 아래 칸을 제외한 칸의 오른쪽, 아래쪽 벽 유무 정보를 저장한 후, 맨 오른쪽 칸과 맨 아래 칸의 정보를 마지막에 한꺼번에 모아서 저장한다.

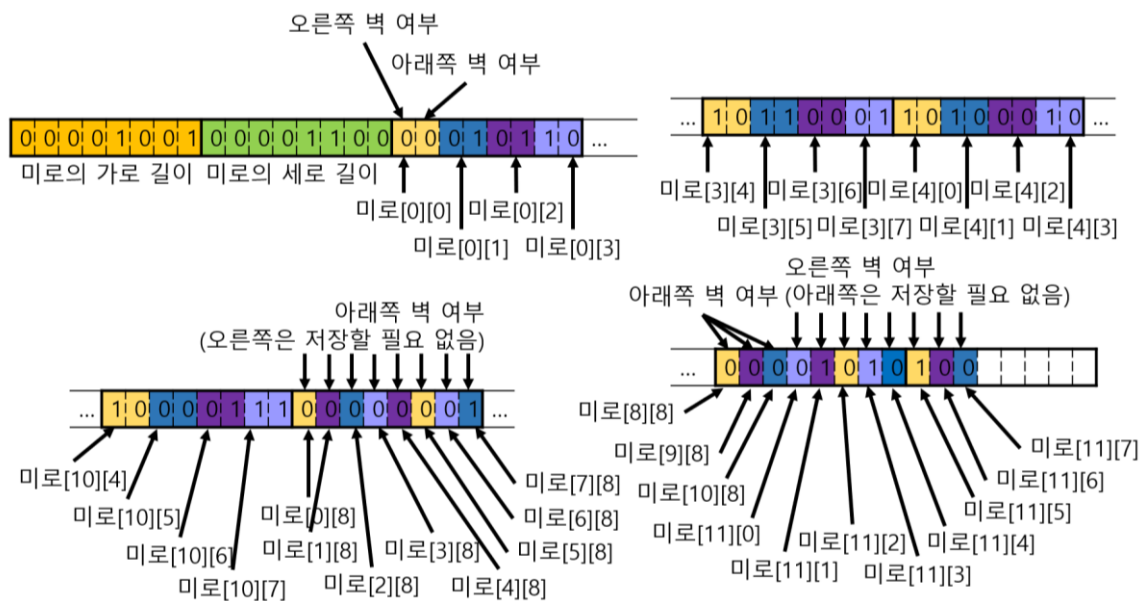


그림 13.1

저장된 미로를 복원하는 것도 미로의 크기를 읽은 후, 그 크기에 맞게 2차원 배열을 동적 할당하고 이후 2비트씩 데이터를 읽어 상하좌우에 벽이 있는지를 저장하는 방식으로 할 수 있다. 2비트에 포함되지 않은 위쪽과 왼쪽의 벽 정보는 위쪽 칸과 왼쪽 칸의 정보를 참고해 복원할 수 있고, 1행 또는 1열인 경우 벽이 있는 것으로 처리하면 된다. 1비트만 사용하는 오른쪽 끝 또는 아래쪽 끝은 1비트씩 읽어 별도로 처리하면 그 부분도 쉽게 복원할 수 있다.

## 14. 소스코드

첨부파일 참조

문제별 코드 실행 시 다음 파일이 모두 필요합니다.

문제 1 코드 실행 시: 문제 1 코드 필요(다른 코드 필요 없음)

문제 3 코드 실행 시: 프로그래밍 과제 3 공통 코드, 프로그래밍 과제 3 공통 헤더, 문제 3 코드 필요

문제 4 코드 실행 시: 프로그래밍 과제 3 공통 코드, 프로그래밍 과제 3 공통 헤더, 문제 4 코드 필요

문제 2, 문제 5는 코드 없음

## 15. 참고문헌

Kaden Sungbin Cho, 「파이썬 리스트 내부구조 (Python List Internals)」, 『tistory』, 2021.02.13  
(<https://kadensungbincho.tistory.com/59>)

seoyeon hwang, 「파이썬 리스트 내부 구조」, 『medium』, 2021.05.04  
(<https://seoyeonhwng.medium.com/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EB%A6%AC%EC%8A%A4%ED%8A%B8-%EB%82%B4%EB%B6%80-%EA%B5%AC%EC%A1%B0-f04847b58286>)

zpoint, 「CPython-Internals/BasicObject/list/list.md」, 『github zpoint/CPython-Internals』,  
2020.07.09 (<https://github.com/zpoint/CPython-Internals/blob/master/BasicObject/list/list.md>)