

# 데이터 구조

**순차 자료구조와  
선형 리스트**

# Contents

## ❖ 학습목표

- 순차 자료구조의 개념과 특징을 알아본다.
- 선형 리스트의 개념과 연산을 알아본다.
- C 언어를 이용해 선형 리스트의 순차 자료구조를 구현한다.
- 선형 리스트의 응용과 순차 자료구조 구현 방법을 알아본다.

## ❖ 내용

- 01 순차 자료구조와 선형 리스트의 이해
- 02 선형 리스트의 연산과 알고리즘
- 03 선형 리스트의 응용 및 구현

# 1. 순차 자료구조와 선형 리스트의 이해

## ❖ 순차 자료구조의 개념

- 구현할 자료들을 논리적 순서로 메모리에 연속 저장하는 구현 방식
- 논리적인 순서와 물리적인 순서가 항상 일치해야 함
- C 프로그래밍에서 순차 자료구조의 구현 방식 제공하는 프로그램 기법은 배열

표 3-1 순차 자료구조와 연결 자료구조의 비교

구분	순차 자료구조	연결 자료구조
메모리 저장 방식	메모리의 저장 시작 위치부터 빈자리 없이 자료를 순서대로 연속하여 저장한다. 논리적인 순서와 물리적인 순서가 일치하는 구현 방식이다.	메모리에 저장된 물리적 위치나 물리적 순서와 상관없이, 링크에 의해 논리적인 순서를 표현하는 구현 방식이다.
연산 특징	삽입·삭제 연산을 해도 빈자리 없이 자료가 순서대로 연속하여 저장된다. 변경된 논리적인 순서와 저장된 물리적인 순서가 일치한다.	삽입·삭제 연산을 하여 논리적인 순서가 변경되어도, 링크 정보만 변경되고 물리적 순서는 변경되지 않는다.
프로그램 기법	배열을 이용한 구현	포인터를 이용한 구현

# 1. 순차 자료구조와 선형 리스트의 이해

## ❖ 선형 리스트의 표현

- 리스트 : 자료를 구조화하는 가장 기본적인 방법은 나열하는 것

표 3-2 리스트 예

동창 이름	좋아하는 음식	오늘 할 일
상원	김치찌개	운동
상범	닭볶음탕	자료구조 스터디
수영	된장찌개	과제 제출
현정	잡채	동아리 공연 연습
...	...	...

# 1. 순차 자료구조와 선형 리스트의 이해

## ❖ 선형 리스트 Linear List

- 순서 리스트 Ordered List
- 자료들 간에 순서를 갖는 리스트

표 3-3 선형 리스트 예

동창 이름		좋아하는 음식		오늘 할 일	
1	상원	1	김치찌개	1	운동
2	상범	2	닭볶음탕	2	자료구조 스터디
3	수영	3	된장찌개	3	과제 제출
4	현정	4	잡채	4	동아리 공연 연습
...		...	...	...	...

# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 리스트의 표현 형식

리스트 이름 = (원소 1, 원소 2, ..., 원소 n)

①

(a) 리스트 표현 형식

그림 3-1 리스트 표현 형식과 예

동창 = (상원, 상범, 수영, 현정)

(b) 리스트 표현 예

공백 리스트 이름 = ( )

그림 3-2 공백 리스트 형식

# 1. 순차 자료구조와 선형 리스트의 이해

## ❖ 선형 리스트의 구현

### ■ 선형 순차 리스트 (Linear Sequential List)

- 원소를 논리적인 순서대로 메모리에 연속하여 저장하는 순차 자료구조 방식의 선형 리스트

### ■ 선형 연결 리스트 (Linear Linked List)

- 원소를 논리적인 순서대로 연결하여 구성하는 연결 자료구조 방식의 선형 리스트

☞ 일반적으로 선형 순차 리스트를 선형 리스트라고 하고,  
선형 연결 리스트는 연결 리스트(Linked List)라고 함



그림 3-3 선형 리스트의 메모리 저장 구조 예

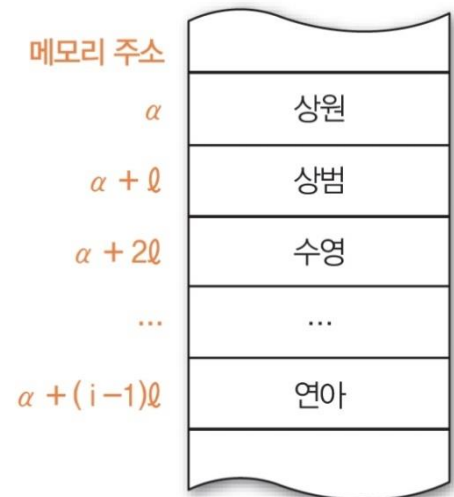


그림 3-4 선형 리스트에서 원소의 위치

# 1. 순차 자료구조와 선형 리스트의 이해

## ❖ 선형 리스트의 배열 표현

- 1차원 배열을 이용한 선형 리스트 표현

표 3-4 분기별 노트북 판매량 리스트

분기	1/4 분기	2/4 분기	3/4 분기	4/4 분기
판매량	157	209	251	312

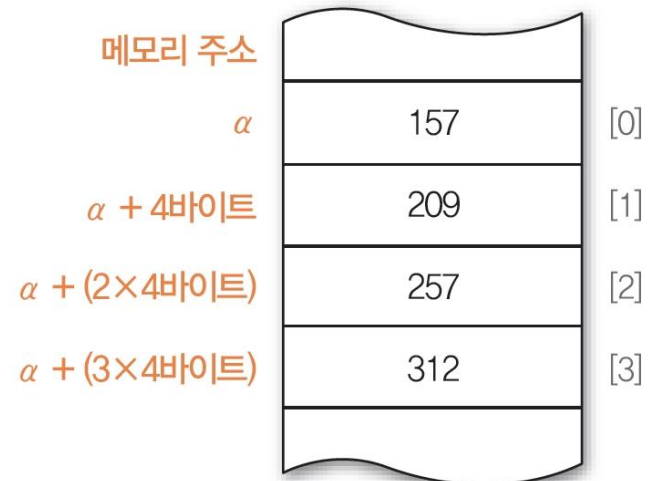
```
int sale[4] = { 157, 209, 251, 312 };
```

(a) 분기별 판매량 선형 리스트의 1차원 배열 선언

	[0]	[1]	[2]	[3]
sale	157	209	251	312

(b) 분기별 판매량 선형 리스트의 논리적 구조

그림 3-5 분기별 판매량 선형 리스트 예

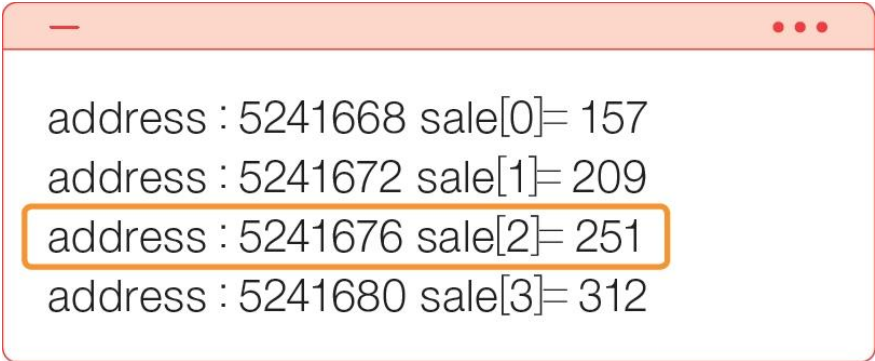


(c) 분기별 판매량 선형 리스트의 물리적 구조



# 1. 순차 자료구조와 선형 리스트의 이해

- [예제 3-1] 원소의 논리적·물리적 순서 확인하기 : [교재 124p](#)



```
address : 5241668 sale[0]= 157
address : 5241672 sale[1]= 209
address : 5241676 sale[2]= 251
address : 5241680 sale[3]= 312
```

- 실행 결과 확인
  - 배열 sale의 시작 주소 : 5241668
  - $\text{sale}[2]=251$ 의 위치 = 시작 주소 + (인덱스  $\times$  4바이트)  
=  $5241668 + (2 \times 4\text{바이트})$   
= 5241676
  - 논리적인 순서대로 메모리에 연속하여 저장된 순차 구조임을 확인!

# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 2차원 배열을 이용한 선형 리스트 표현

표 3-5 2021~2022년 분기별 노트북 판매량 리스트

연도 \ 분기	1/4분기	2/4분기	3/4분기	4/4분기
2021년	63	84	140	130
2022년	157	209	251	312

```
int sale[2][4] = { { 63, 84, 140, 130 },  
                  { 157, 209, 251, 312 } };
```

	[0]	[1]	[2]	[3]
sale [0]	63	84	140	130
[1]	157	209	251	312

그림 3-6 2021~2022년 분기별 판매량 선형 리스트의 예

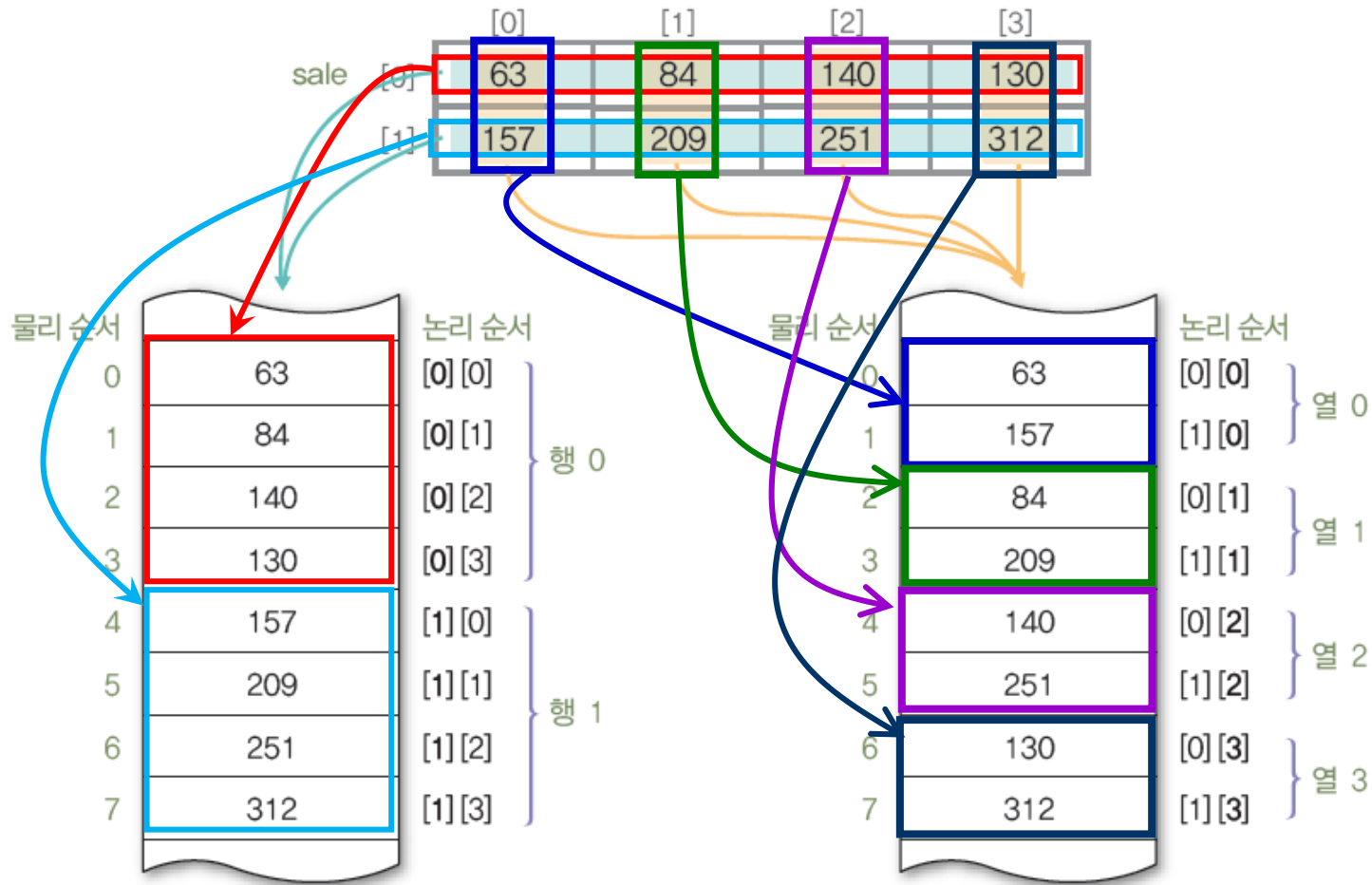
# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 2차원 배열의 물리적 저장 방법

- 2차원의 논리적 순서를 1차원의 물리적 순서로 변환하는 방법 사용
- 행 우선 순서 방법 row major order
  - 2차원 배열의 첫 번째 인덱스인 행 번호를 기준으로 사용하는 방법
  - $\text{sale}[0][0]=63, \text{sale}[0][1]=84, \text{sale}[0][2]=140, \text{sale}[0][3]=130, \text{sale}[1][0]=157, \text{sale}[1][1]=209, \text{sale}[1][2]=251, \text{sale}[1][3]=312$
  - 원소의 위치 계산 방법 :  $\alpha + (i \times n_j + j) \times \ell$   
행의 개수가  $n_i$ 이고 열의 개수가  $n_j$ 인 2차원 배열  $A[n_i][n_j]$ 의 시작주소가  $\alpha$ 이고 원소의 길이가  $\ell$  일 때,  $i$ 행  $j$ 열 원소 즉,  $A[i][j]$ 의 위치
- 열 우선 순서 방법 column major order
  - 2차원 배열의 마지막 인덱스인 열 번호를 기준으로 사용하는 방법
  - $\text{sale}[0][0]=63, \text{sale}[1][0]=157, \text{sale}[0][1]=84, \text{sale}[1][1]=209, \text{sale}[0][2]=140, \text{sale}[1][2]=251, \text{sale}[0][3]=130, \text{sale}[1][3]=312$
  - 원소의 위치 계산 방법 :  $\alpha + (j \times n_i + i) \times \ell$

# 1. 순차 자료구조와 선형 리스트의 이해

- 2차원 논리 순서를 1차원 물리 순서로 변환



(a) 행 우선 순서 방법

(b) 열 우선 순서 방법

그림 3-11 2차원 논리 순서를 1차원 물리 순서로 변환

# 1. 순차 자료구조와 선형 리스트의 이해

- [예제 3-2] 2차원 배열의 논리적·물리적 순서 확인하기 : [교재 126p](#)



```
address: 1374812 sale 0 = 63
address: 1374816 sale 1 = 84
address: 1374820 sale 2 = 140
address: 1374824 sale 3 = 130
address: 1374828 sale 4 = 157
address: 1374832 sale 5 = 209
address: 1374836 sale 6 = 251
address: 1374840 sale 7 = 312
```

- 실행 결과 확인

- 시작 주소  $\alpha = 13474812$ ,  $n_i = 2$ ,  $n_j = 4$ ,  $i = 1$ ,  $j = 1$ ,  $\ell = 4$
- $\text{sale}[1][2] = 251$ 의 위치  $= \alpha + (i \times n_j + j) \times \ell$ 
$$= 13474812 + (1 \times 4 + 2) \times 4$$
$$= 13474812 + 24$$
$$= 13474836$$
- C 컴파일러가 행 우선 순서 방법으로 2차원 배열을 저장함을 확인!

# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 3차원 배열을 이용한 선형 리스트 표현

표 3-6 1팀과 2팀의 분기별 노트북 판매량 리스트

팀	연도 \ 분기	1/4분기	2/4분기	3/4분기	4/4분기
1팀	2021년	63	84	140	130
	2022년	157	209	251	312
2팀	2021년	59	80	130	135
	2022년	149	187	239	310

```
int sale[2][2][4] = { { { 63, 84, 140, 130 },  
                        { 157, 209, 251, 312 } },  
                      { { 59, 80, 130, 135 },  
                        { 149, 187, 239, 310 } }  
};
```

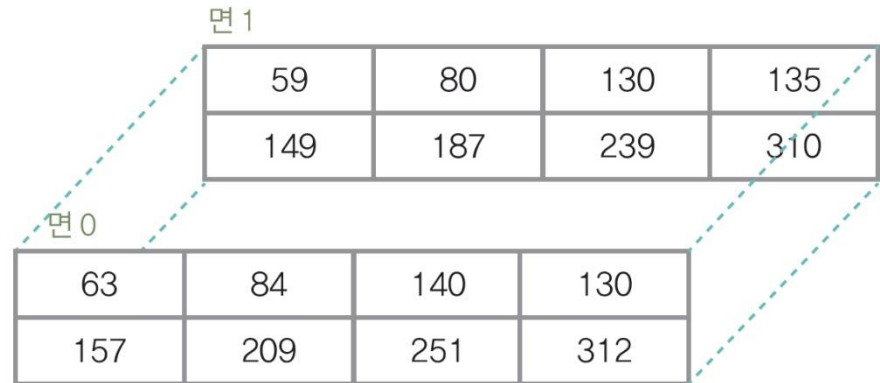


그림 3-8 선형 리스트의 3차원 배열 예

# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 3차원 배열의 물리적 저장 방법

- 3차원의 논리적 순서를 1차원의 물리적 순서로 변환하는 방법 사용
- 면 우선 순서 방법

- 3차원 배열의 첫 번째 인덱스인 면 번호를 기준으로 사용하는 방법

- 원소의 위치 계산 방법 :  $\alpha + \{(i \times n_j \times n_k) + (j \times n_k) + k\} \times \ell$

면의 개수가  $n_i$ 이고 행의 개수가  $n_j$ 이고, 열의 개수가  $n_k$  인 3차원 배열  $A[n_i][n_j][n_k]$ , 시작주소가  $\alpha$ 이고 원소의 길이가  $\ell$  일 때,  $i$ 면  $j$ 행  $k$ 열 원소 즉,  $A[i][j][k]$ 의 위치

- 열 우선 순서 방법

- 3차원 배열의 마지막 인덱스인 열 번호를 기준으로 사용하는 방법

- 원소의 위치 계산 방법 :  $\alpha + \{(k \times n_j \times n_i) + (j \times n_i) + i\} \times \ell$

# 1. 순차 자료구조와 선형 리스트의 이해

## ■ 3차원의 논리 순서에 대한 1차원의 물리 순서 변환

물리 순서		논리 순서	
0	63	[0][0][0]	면 0
1	84	[0][0][1]	
2	140	[0][0][2]	
3	130	[0][0][3]	
4	157	[0][1][0]	
5	209	[0][1][1]	면 1
6	251	[0][1][2]	
7	312	[0][1][3]	
8	59	[1][0][0]	
9	80	[1][0][1]	
10	130	[1][0][2]	면 2
11	135	[1][0][3]	
12	149	[1][1][0]	
13	187	[1][1][1]	
14	239	[1][1][2]	
15	310	[1][1][3]	면 3

(a) 면 우선 방법

물리 순서		논리 순서	
0	63	[0][0][0]	열 0
1	59	[1][0][0]	
2	157	[0][1][0]	
3	149	[1][1][0]	열 1
4	84	[0][0][1]	
5	80	[1][0][1]	
6	209	[0][1][1]	열 2
7	187	[1][1][1]	
8	140	[0][0][2]	
9	130	[1][0][2]	열 3
10	251	[0][1][2]	
11	239	[1][1][2]	
12	130	[0][0][3]	열 4
13	135	[1][0][3]	
14	312	[0][1][3]	
15	310	[1][1][3]	열 5

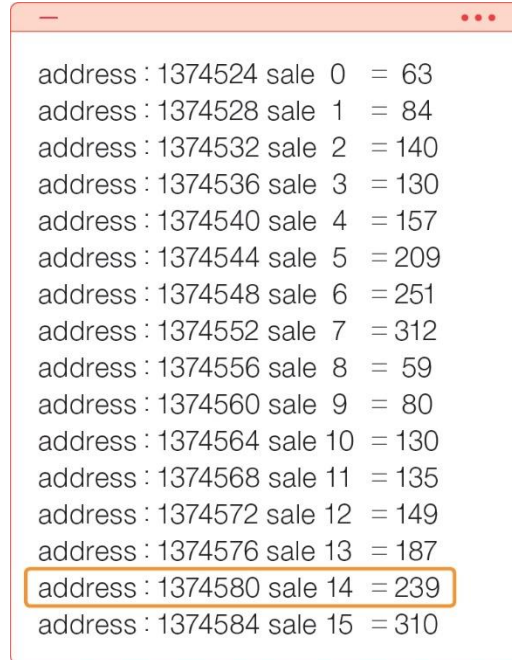
(b) 열 우선 방법

그림 3-9 3차원의 논리 순서에 대한 1차원의 물리 순서 변환



# 1. 순차 자료구조와 선형 리스트의 이해

- [예제 3-3] 3차원 배열의 논리적·물리적 순서 확인하기 : [교재 129p](#)



address : 1374524	sale 0	= 63
address : 1374528	sale 1	= 84
address : 1374532	sale 2	= 140
address : 1374536	sale 3	= 130
address : 1374540	sale 4	= 157
address : 1374544	sale 5	= 209
address : 1374548	sale 6	= 251
address : 1374552	sale 7	= 312
address : 1374556	sale 8	= 59
address : 1374560	sale 9	= 80
address : 1374564	sale 10	= 130
address : 1374568	sale 11	= 135
address : 1374572	sale 12	= 149
address : 1374576	sale 13	= 187
address : 1374580	sale 14	= 239
address : 1374584	sale 15	= 310

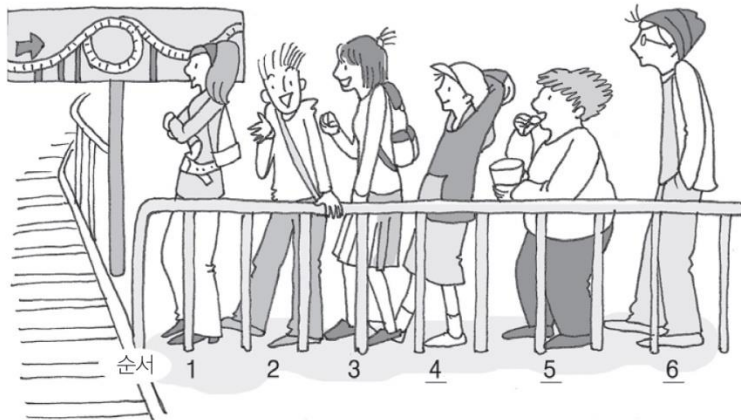
- 실행 결과 확인

- 시작 주소  $\alpha=1374524$ ,  $n_i=2$ ,  $n_j=2$ ,  $n_k=4$ ,  $i=1$ ,  $j=1$ ,  $k=2$ ,  $\ell=4$
- $\text{sale}[1][1][2]=239$ 의 위치  $= \alpha + \{(i \times n_j \times n_k) + (j \times n_k) + k\} \times \ell$ 
$$= 1374524 + \{(1 \times 2 \times 4) + (1 \times 4) + 2\} \times 4$$
$$= 1374524 + 56$$
$$= 1374580$$
- C 컴파일러가 면 우선 순서 방법으로 3차원 배열을 저장함을 확인!

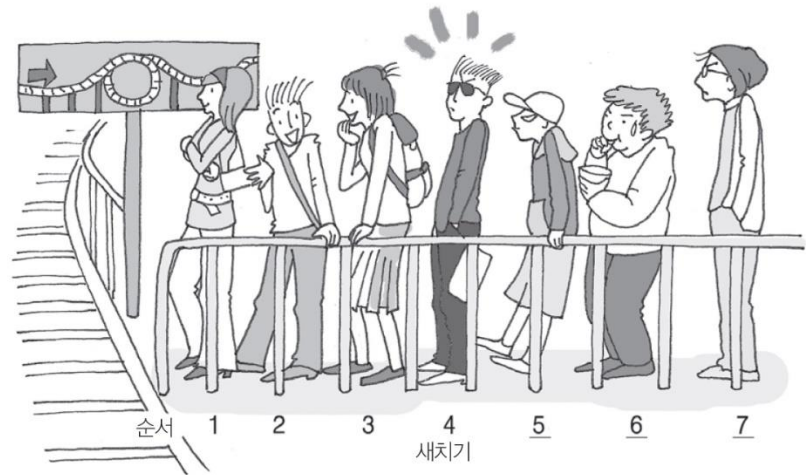
## 2. 선형 리스트의 연산과 알고리즘

### ❖ 선형 리스트에서 원소 삽입과 알고리즘

- 선형리스트 중간에 원소가 삽입되면, 그 이후의 원소들은 한 자리씩 자리를 뒤로 이동하여 물리적 순서를 논리적 순서와 일치시킴



(a) 새치기 전



(b) 새치기 후

그림 3-10 새치기 전과 후의 위치와 순서 변화

## 2. 선형 리스트의 연산과 알고리즘

### ■ 원소 삽입 방법

#### ① 원소를 삽입할 빈 자리 만들기

☞ 삽입할 자리 이후의 원소들을 한 자리씩 뒤로 자리 이동

- 자리 이동하면서 원소가 덮어쓰기 되지 않도록, 마지막 원소부터 자리 이동(①~④)

#### ② 준비한 빈 자리에 원소 삽입하기



그림 3-11 선형 리스트에서의 원소 삽입

## 2. 선형 리스트의 연산과 알고리즘

- 삽입할 자리를 만들기 위한 자리 이동 횟수
  - (n+1)개의 원소로 이루어진 선형 리스트에서 k번 자리에 원소를 삽입하는 경우 : k번 원소부터 마지막 인덱스 n번 원소까지 (n-k+1)개의 원소를 이동
    - 이동횟수 =  $n-k+1$  = **마지막 원소의 인덱스 - 삽입할 자리의 인덱스 + 1**
- [알고리즘 3-1] 선형 리스트의 원소 삽입

### 알고리즘 3-1 선형 리스트의 원소 삽입

```
insertElement(L, n, x)
  ① for (i ← 0; i < n; i ← i + 1) do {
    ①-a if (L[i] <= x && x <= L[i+1]) then {
      ①-b k ← i + 1;
      break;
    }
  }
  ② if (i == n) then k ← n;
  ③ for (i ← n; i > k; i ← i - 1) do // n-1번부터 k번까지
    L[i] ← L[i-1];                // 뒤로 한 자리 이동
  ④ L[k] ← x;
end insertElement()
```

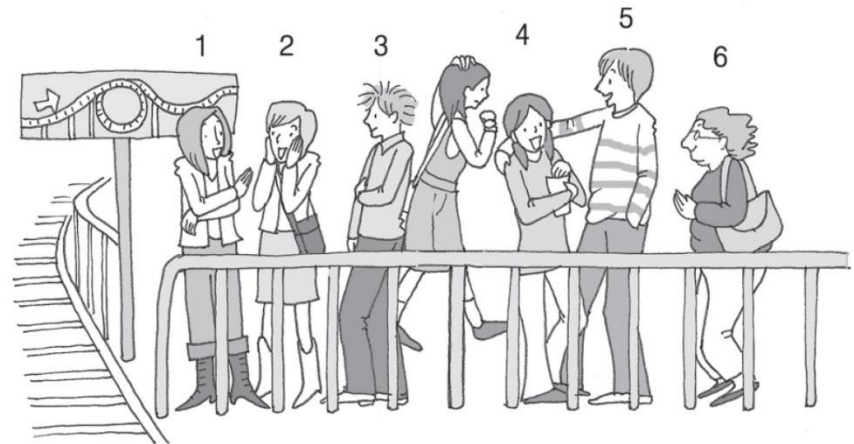
## 2. 선형 리스트의 연산과 알고리즘

### ❖ 선형 리스트에서 원소 삭제와 알고리즘

- 선형리스트 중간에서 원소가 삭제되면, 그 이후의 원소들은 한 자리씩 자리를 앞으로 이동하여 물리적 순서를 논리적 순서와 일치시킴



(a) 4번이 줄에서 나가기 전



(b) 원래 4번이 줄에서 나간 후

그림 3-12 줄에서 사람이 나간 후의 위치와 순서 변화

## 2. 선형 리스트의 연산과 알고리즘

### ■ 원소 삭제 방법

#### ① 원소 삭제하기

#### ② 삭제한 빈 자리 채우기

☞ 삭제한 자리 이후의 원소들을 한자리씩 앞으로 자리 이동

- 자리 이동하면서 원소가 덮어쓰기 되지 않도록, 앞에서부터 자리 이동(①~④)



그림 3-13 선형 리스트에서의 원소 삭제



## 2. 선형 리스트의 연산과 알고리즘

### ❖ 선형 리스트 프로그램

- [예제 3-4] 선형 리스트에 원소 삽입하고 삭제하기 : [교재 134p](#)

삽입 전 선형 리스트 : 10 20 40 50 60 70

원소의 갯수 : 6

삽입 후 선형 리스트 : 10 20 30 40 50 60 70

원소의 갯수 : 7

자리 이동 횟수 : 4

삭제 후 선형 리스트 : 10 20 40 50 60 70

원소의 갯수 : 6

자리 이동 횟수 : 4



### 3. 선형 리스트의 응용 및 구현

#### ❖ 행렬의 선형 리스트 표현

##### ■ 행렬matrix의 개념

- 행과 열로 구성된 자료구조
- $m \times n$  행렬 : 행 개수가  $m$ 개, 열 개수가  $n$ 개인 행렬
- 정방행렬 : 행렬 중에서  $m$ 과  $n$ 이 같은 행렬
- 전치행렬 : 행렬의 행과 열을 서로 바꿔 구성한 행렬

$$A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{vmatrix}$$

$$A' = \begin{vmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{vmatrix}$$

(a) 행렬  $A(m \times n$  행렬)와 전치행렬  $A'(n \times m$  행렬) 표현

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix} \xrightarrow{\text{전치행렬로 변환}} A' = \begin{vmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{vmatrix}$$

(b)  $3 \times 4$  행렬  $A$ 와 전치행렬  $A'$  예


### 3. 선형 리스트의 응용 및 구현

$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix}$  

$A[3][4]$

	[0]	[1]	[2]	[3]
[0]	1	2	3	4
[1]	5	6	7	8
[2]	9	10	11	12

그림 3-15 행렬 A의 2차원 배열 표현 예

$B = \begin{vmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 7 & 0 & 0 \\ 23 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 31 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 & 0 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 \\ 52 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11 & 0 & 0 \end{vmatrix}$  

$B[8][7]$

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]	0	0	2	0	0	0	12
[1]	0	0	0	0	7	0	0
[2]	23	0	0	0	0	0	0
[3]	0	0	0	31	0	0	0
[4]	0	14	0	0	0	25	0
[5]	0	0	0	0	0	0	6
[6]	52	0	0	0	0	0	0
[7]	0	0	0	0	11	0	0

그림 3-16 희소행렬 B의 2차원 배열 표현 예 1

희소행렬 Sparse Matrix은 실제로 사용하지 않는 공간이 많아  
기억 공간의 활용도가 떨어짐

### 3. 선형 리스트의 응용 및 구현

---

- 희소 행렬에 대한 2차원 배열 표현
  - 희소 행렬 B는 배열의 원소 56개 중 실제 사용하는 것은 0이 아닌 원소를 저장하는 10개뿐이므로 46개의 메모리 공간 낭비
    - ① 0이 아닌 원소만 추출하여 <행번호, 열번호, 원소> 쌍으로 배열에 저장
    - ② 추출한 순서쌍을 2차원 배열에 행으로 저장
    - ③ 원래의 행렬에 대한 정보를 순서쌍으로 작성하여 0번 행에 저장

### 3. 선형 리스트의 응용 및 구현

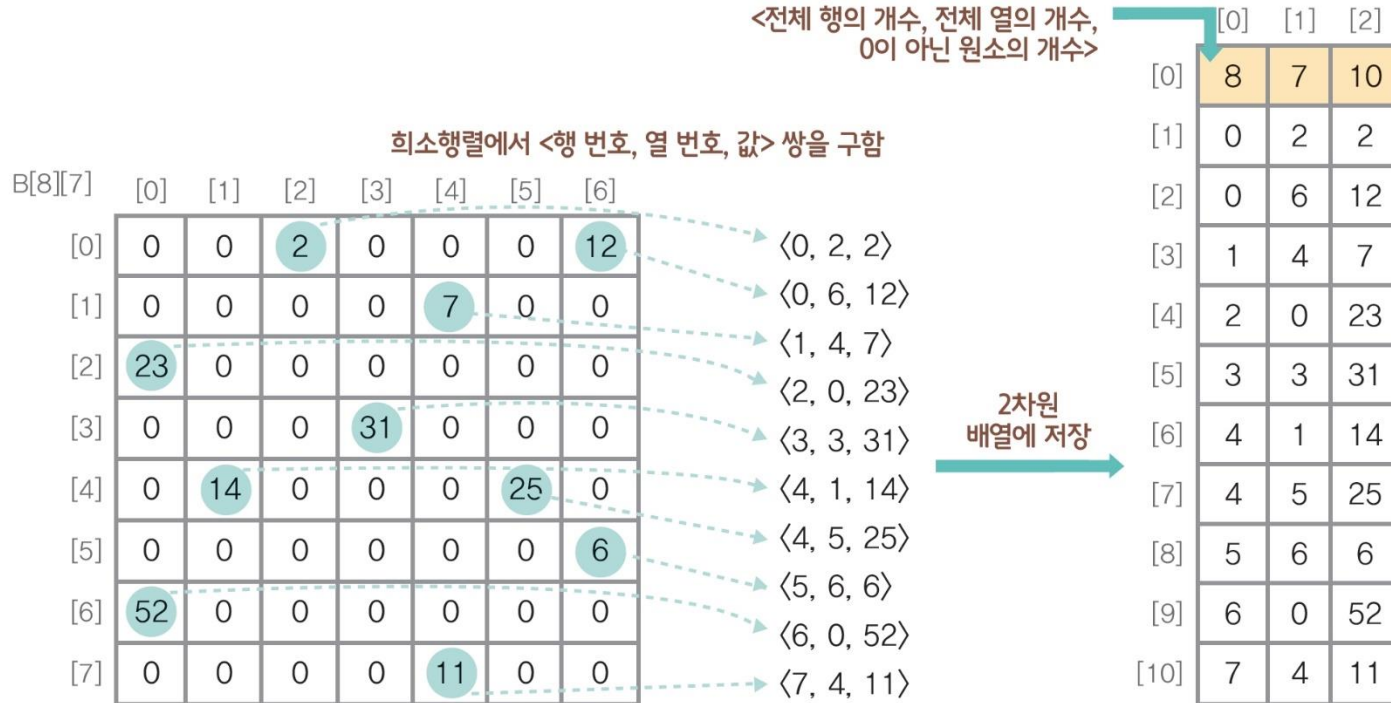


그림 3-17 희소행렬 B의 2차원 배열 표현 예 2

기억 공간을 좀 더 효율적으로 사용하기 위해 0이 아닌 값이 있는  
행렬 원소만 따로 배열로 구성하는 방법 사용

### 3. 선형 리스트의 응용 및 구현

#### ■ [ADT 3-1] 희소행렬의 추상 자료형

##### ADT 3-1    희소행렬의 추상 자료형

ADT Sparse\_Matrix

데이터 : 3원소 쌍 <행 인덱스, 열 인덱스, 원소값>의 집합

// a, b는 희소행렬, c는 행렬, u와 v는 행렬의 원소값, i와 j는 행 인덱스와 열 인덱스

연산 : a, b ∈ Sparse\_Matrix; c ∈ Matrix; u, v ∈ value; i ∈ Row; j ∈ Column;

//  $m \times n$ 의 공백 희소행렬을 만드는 연산

smCreate(m, n) ::= return an empty sparse matrix with  $m \times n$ ;

// 희소행렬 a[i, j]=v를 c[j, i]=v로 전치시킨 전치행렬 c를 구하는 연산

smTranspose(a) ::= return c where c[j, i] = v when a[i, j] = v;

// 차수가 같은 희소행렬 a와 b를 합한 행렬 c를 구하는 연산

smAdd(a, b) ::= if (a.dimension = b.dimension)  
                  then return c where c[i, j] = v + u where a[i, j] = v and b[i, j] = u  
                  else return error;

// 희소행렬 a의 열(n)과 희소행렬 b의 행(m) 개수가 같은 경우에 두 행렬의 곱을 구하는 연산

smMulti(a, b) ::= if (a.n = b.m) then return c where c[i, j] = a[i, k] × b[k, j];  
                  else return error;

End Sparse\_Matrix

### 3. 선형 리스트의 응용 및 구현

#### ■ [알고리즘 3-3] 희소행렬의 전치 연산

알고리즘 3-3    희소행렬의 전치 연산

```
smTranspose(a[])
  m ← a[0, 0];      // 희소행렬 a의 행 수
  n ← a[0, 1];      // 희소행렬 a의 열 수
  v ← a[0, 2];      // 희소행렬 a에서 0이 아닌 원소 수
  b[0, 0] ← n;      // 전치행렬 b의 행 수 지정
  b[0, 1] ← m;      // 전치행렬 b의 열 수 지정
  b[0, 2] ← v;      // 전치행렬 b의 원소 수 지정
  if (v > 0) then { // 0이 아닌 원소가 있는 경우에만 전치 연산 수행
    p ← 1;
    for (i ← 0; i < n; i ← i + 1) do { // 희소행렬 a의 열별로 전치 반복 수행
      for (j ← 1; j ≤ v; j ← j + 1) do { // 0이 아닌 원소 수에 대해서만 반복 수행
        if (a[j, 1] = i) then { // 현재의 열에 속하는 원소가 있으면 b[]에 삽입
          b[p, 0] ← a[j, 1];
          b[p, 1] ← a[j, 0];
          b[p, 2] ← a[j, 2];
          p ← p + 1;
        }
      }
    }
  }
  return b[];
end smTranspose()
```

### 3. 선형 리스트의 응용 및 구현

- [예제 3-5] 희소행렬의 전치 연산하기 : [교재 140p](#)

⟨⟨희소행렬 a⟩⟩

[ 8, 7, 10 ]

[ 0, 2, 2 ]

[ 0, 6, 12 ]

[ 1, 4, 7 ]

[ 2, 0, 23 ]

[ 3, 3, 31 ]

[ 4, 1, 14 ]

[ 4, 5, 25 ]

[ 5, 6, 6 ]

[ 6, 0, 52 ]

[ 7, 4, 11 ]

⟨⟨희소행렬 b⟩⟩

[ 7, 8, 10 ]

[ 0, 2, 23 ]

[ 0, 6, 52 ]

[ 1, 4, 14 ]

[ 2, 0, 2 ]

[ 3, 3, 31 ]

[ 4, 1, 7 ]

[ 4, 7, 11 ]

[ 5, 4, 25 ]

[ 6, 0, 12 ]

[ 6, 5, 6 ]

### 3. 선형 리스트의 응용 및 구현

#### ❖ 다항식의 선형 리스트 표현

##### ■ 다항식의 개념

- $aX^e$  형식의 항들의 합으로 구성된 식
  - $a$  : 계수(coefficient)
  - $X$  : 변수(variable)
  - $e$  : 지수(exponent)

##### ■ 다항식의 특징

- 지수에 따라 내림차순으로 항을 나열
- 다항식의 차수 : 가장 큰 지수
- 다항식 항의 최대 개수 = (차수 + 1)개

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

그림 3-18  $n$ 차 다항식  $P(x)$



# 3. 선형 리스트의 응용 및 구현

## ■ [ADT 3-2] 다항식의 추상 자료형

### ADT 3-2 다항식의 추상 자료형

ADT Polynomial

데이터 : 지수( $e_i$ ) - 계수( $a_i$ )의 순서쌍  $\langle e_i, a_i \rangle$ 의 집합으로 표현된 다항식

$p(x) = a_0x^{e_0} + a_1x^{e_1} + \dots + a_ix^{e_i} + \dots + a_nx^{e_n}$ . ( $e_i$ 는 음이 아닌 정수)

//  $p$ ,  $p1$ ,  $p2$ 는 다항식,  $a$ 는 계수,  $e$ 는 지수

연산 :  $p, p1, p2 \in \text{Polynomial}$ ;  $a \in \text{Coefficient}$ ;  $e \in \text{Exponent}$ ;

// 공백 다항식  $p(x) = 0$ 을 만드는 연산

$\text{zeroP}() ::= \text{return polynomial } p(x) = 0;$

// 다항식  $p$ 가 0인지 검사하여 0이면 true를 반환하는 연산

$\text{isZeroP}(p) ::= \text{if } (p) \text{ then false}$   
 $\quad \text{else return true};$

// 다항식  $p$ 에서 지수가  $e$ 인 항의 계수  $a$ 를 구하는 연산

$\text{coef}(p, e) ::= \text{if } \langle e, a \rangle \in p \text{ then return } a$   
 $\quad \text{else return } 0;$

// 다항식  $p$ 에서 최대 지수를 구하는 연산

$\text{maxExp}(p) ::= \text{return max}(p.\text{Exponent});$

// 다항식  $p$ 에 지수가  $e$ 인 항이 없는 경우에 새로운 항  $\langle e, a \rangle$ 를 추가하는 연산

$\text{addTerm}(p, a, e) ::= \text{if } (e \in p.\text{Exponent}) \text{ then return error}$   
 $\quad \text{else return } p \text{ after inserting the term } \langle e, a \rangle;$

// 다항식  $p$ 에서 지수가  $e$ 인 항  $\langle e, a \rangle$ 를 삭제하는 연산

$\text{delTerm}(p, e) ::= \text{if } (e \in p.\text{Exponent}) \text{ then return } p \text{ after removing the term } \langle e, a \rangle$   
 $\quad \text{else return error};$

// 다항식  $p$ 의 모든 항에  $ax^e$ 항을 곱하는 연산

$\text{multTerm}(p, a, e) ::= \text{return } (p * ax^e);$

// 다항식  $p1$ 과  $p2$ 의 합을 구하는 연산

$\text{addPoly}(p1, p2) ::= \text{return } (p1 + p2);$

// 다항식  $p1$ 과  $p2$ 의 곱을 구하는 연산

$\text{multPoly}(p1, p2) ::= \text{return } (p1 * p2);$

End Polynomial

### 3. 선형 리스트의 응용 및 구현

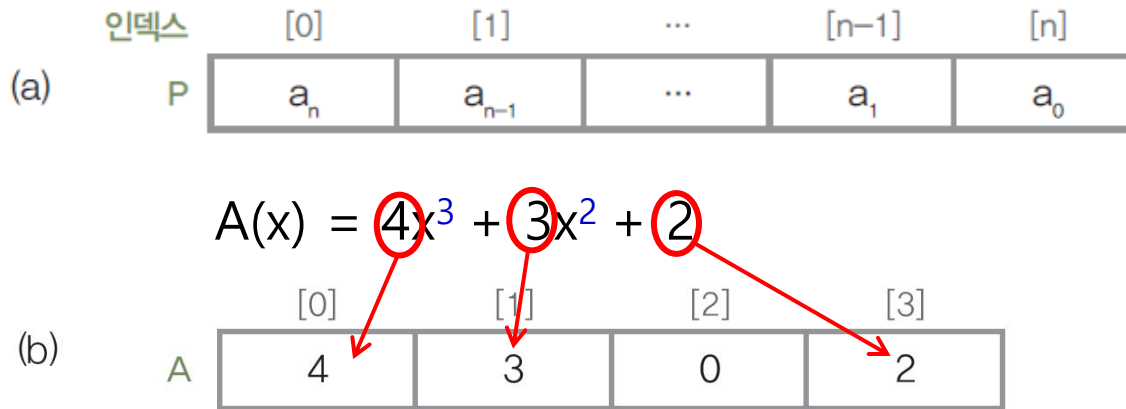


그림 3-19  $n$ 차 다항식  $P(x)$ 의 순차 자료구조 표현과 예

	[0]	[1]	[2]	[3]	...	[997]	[998]	[999]	[1000]
B	3	0	0	0	...	0	0	1	4

(a) 1차원 배열을 이용한 순차 자료구조

	[0]	[1]	
[0]	1000	3	$\rightarrow 3x^{1000}$
[1]	1	1	$\rightarrow x$
[2]	0	4	$\rightarrow 4$

차수가 1000이므로 크기가 1001인 배열을 사용하는데, 항이 3개 뿐이므로 배열의 원소 중에서 3개만 사용

☞ 998개의 배열 원소에 대한 메모리 공간 낭비

(b) 2차원 배열을 이용한 순차 자료구조

그림 3-20 희소 다항식의 순차 자료구조 표현 예 :  $B(x) = 3x^{1000} + x + 4$

### 3. 선형 리스트의 응용 및 구현

#### ■ 다항식의 덧셈 연산

$$\begin{aligned} A(x) &= 4x^3 + 3x^2 + 5x \\ B(x) &= 3x^4 + x^3 + 2x + 1 \\ C(x) &= 3x^4 + (4+1)x^3 + 3x^2 + (5+2)x + 1 \\ &= 3x^4 + 5x^3 + 3x^2 + 7x + 1 \end{aligned}$$

그림 3-21 다항식의 덧셈 addPoly()

### 3. 선형 리스트의 응용 및 구현

#### ■ [알고리즘 3-4] 순차 리스트를 이용한 다항식의 덧셈 연산

알고리즘 3-4 순차 리스트를 이용한 다항식의 덧셈 연산

```
addPoly(A, B)
// 주어진 두 다항식 A와 B를 더하여 결과 다항식 C를 반환하는 알고리즘
C ← zeroP();
while (not isZeroP(A) and not isZeroP(B)) do {
  case {
    maxExp(A) < maxExp(B) :
      C ← addTerm(C, coef(B, maxExp(B)), maxExp(B));


      B ← delTerm(B, maxExp(B));
    maxExp(A) == maxExp(B) :
      sum ← coef(A, maxExp(A)) + coef(B, maxExp(B));
      if (sum ≠ 0) then C ← addTerm(C, sum, maxExp(A));
      A ← delTerm(A, maxExp(A));
      B ← delTerm(B, maxExp(B));
    maxExp(A) > maxExp(B) :
      C ← addTerm(C, coef(A, maxExp(A)), maxExp(A));
      A ← delTerm(A, maxExp(A));
  }
}

if (not isZeroP(A)) then A의 나머지 항들을 C에 복사
else if (not isZeroP(B)) then B의 나머지 항들을 C에 복사
return C;

end addPoly()
```

### 3. 선형 리스트의 응용 및 구현

- [예제 3-6] 순차 리스트를 이용해 다항식 덧셈하기 : [교재 145p](#)



```
A(x)= 4x^3 + 3x^2 + 5x^1 + 0x^0  
B(x)= 3x^4 + 1x^3 + 0x^2 + 2x^1 + 1x^0  
C(x)= 3x^4 + 5x^3 + 3x^2 + 7x^1 + 1x^0
```