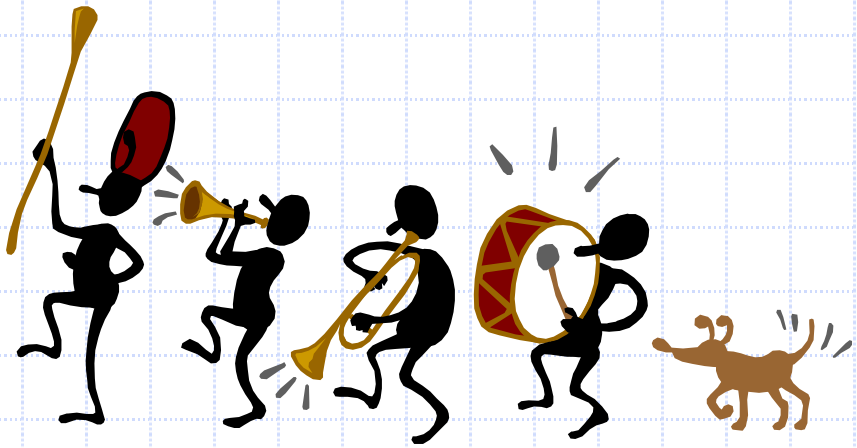
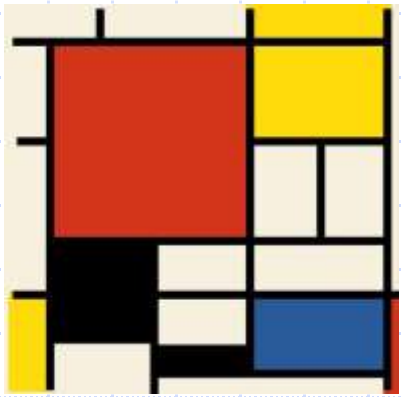
리스트



Outline

- 4.1 추상자료형
- 4.2 리스트 ADT
- 4.3 리스트 ADT 구현
- 4.4 리스트 ADT 확장
- 4.5 응용문제

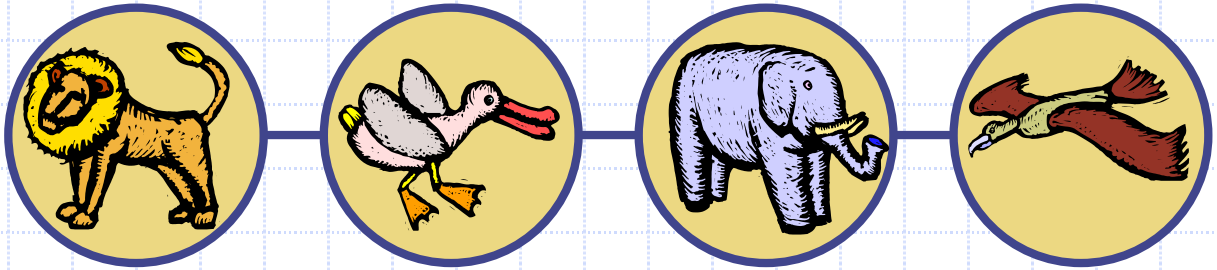
추상자료형



- **추상자료형**(abstract data type, **ADT**): 데이터구조의 추상형
- ADT는 다음을 명세
  - 저장된 데이터
  - 데이터에 대한 작업들
  - 작업 중 발생 가능한 에러 상황들
- **예:** 간단한 주식거래 시스템을 모델링한 ADT
  - 데이터: buy/sell 주문들
  - 지원하는 작업들
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - 에러 상황들
    - 존재하지 않는 주식에 대한 buy/sell 주문
    - 존재하지 않는 주문에 대한 cancel

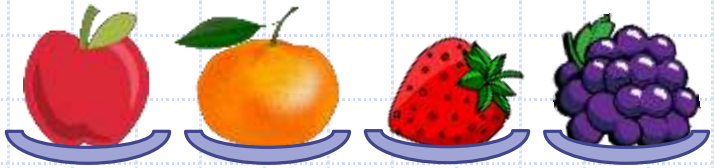
리스트 ADT

- 리스트 ADT는 연속적인 임의 개체들을 모델링
- 원소(element)에 대한 접근 도구
  - **순위**(rank)



리스트 ADT 메쏘드

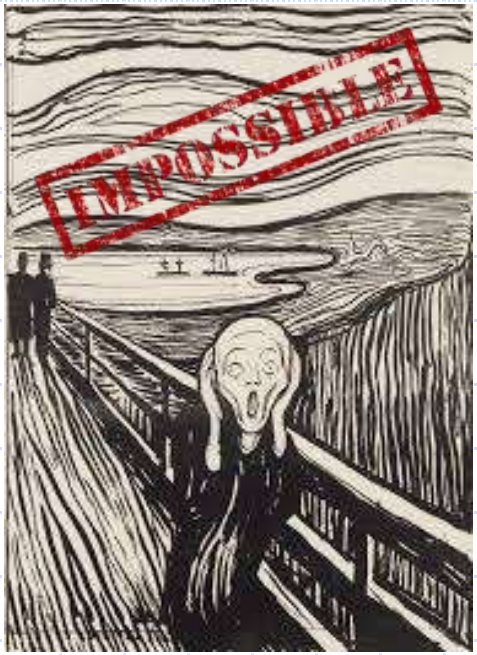
- 원소는 그 **순위**(rank), 즉, 그 원소 앞의 원소 개수를 특정함으로써 접근, 삽입, 또는 삭제



- 일반 메쏘드
  - boolean isEmpty()
  - integer size()
  - iterator elements()
- 접근 메쏘드
  - element get(r)
- 갱신 메쏘드
  - element set(r, e)
  - add(r, e), addFirst(e), addLast(e)
  - element remove(r), element removeFirst(), element removeLast()

예외

- **예외**(exception): 어떤 ADT 작업을 실행하고자 할 때 발생할 수도 있는 오류 상황



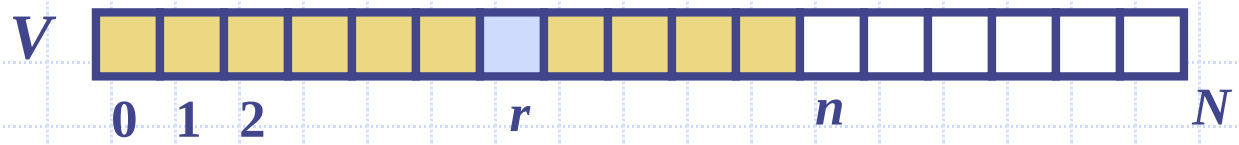
- "실행 불가능한 작업 때문에 예외를 **발령한다**(throw)"고 말한다
- 리스트 ADT에서 발령 가능한 예외들
  - invalidRankException()
  - fullListException()
  - emptyListException()

리스트 응용

- 리스트 ADT: 원소들의 순서(ordered) 집단을 저장하기 위한 기초적이고, 일반적 목적의 데이터구조
- 직접 응용
  - 스택, 큐, 집합 등을 표현하기 위한 도구
  - 소규모 데이터베이스 (**예:** 주소록)
- 간접 응용
  - 더 복잡한 데이터구조를 구축하기 위한 재료로 사용

배열을 이용한 구현

- N개의 단순 또는 복잡한 원소들로 구성된 배열 V
- 변수 n으로 리스트의 크기, 즉 저장된 원소 개수를 관리
- 배열에서 **순위**는 0에서 출발
- 작업 get(r) 또는 set(r, e)는 $O(1)$ 시간에 $V[r]$을 각각 반환 또는 저장하도록 구현
  - $r < 0$ 또는 $r > n - 1$ 인 경우 예외처리 필요

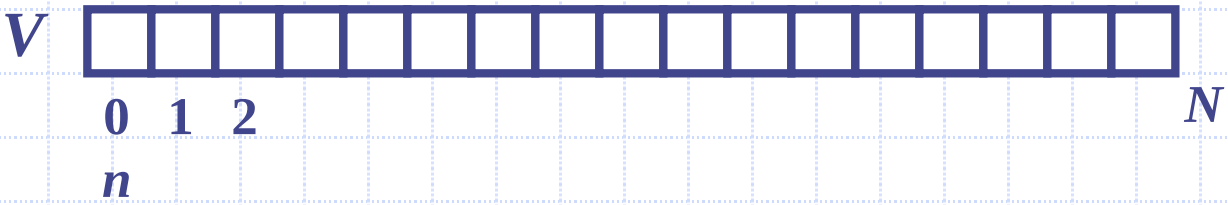


초기화(initialization)

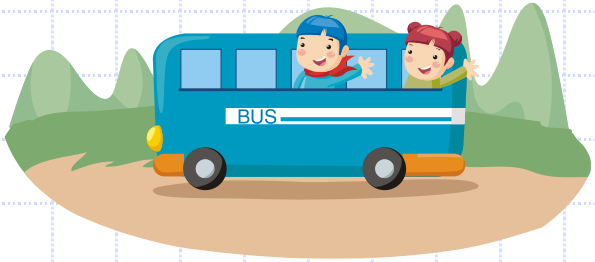
- 초기에는 아무 원소도 없다
- **O**(1) 시간 소요

Alg ***initialize***()
input array ***V***, integer ***N***, ***n***
output an empty array ***V*** of size ***n***

1. ***n*** ← 0
2. **return**



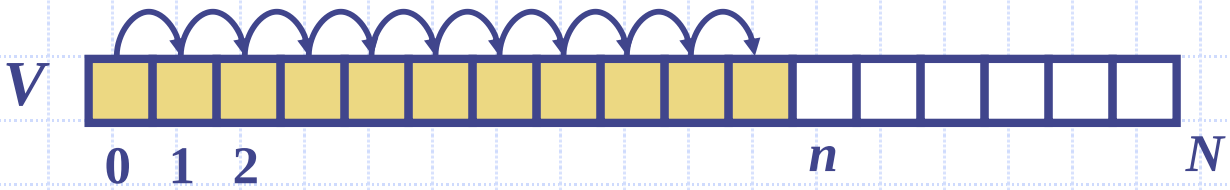
순회(traversal)



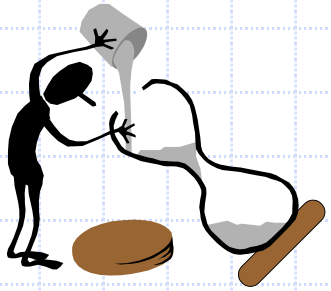
- 작업 traverse는 array의 모든 원소 $V[0]$, $V[1]$, $V[2]$, …, $V[n-1]$을 방문
- $O(n)$ 시간 소요

Alg ***traverse***()
input array V, integer N, n
output none

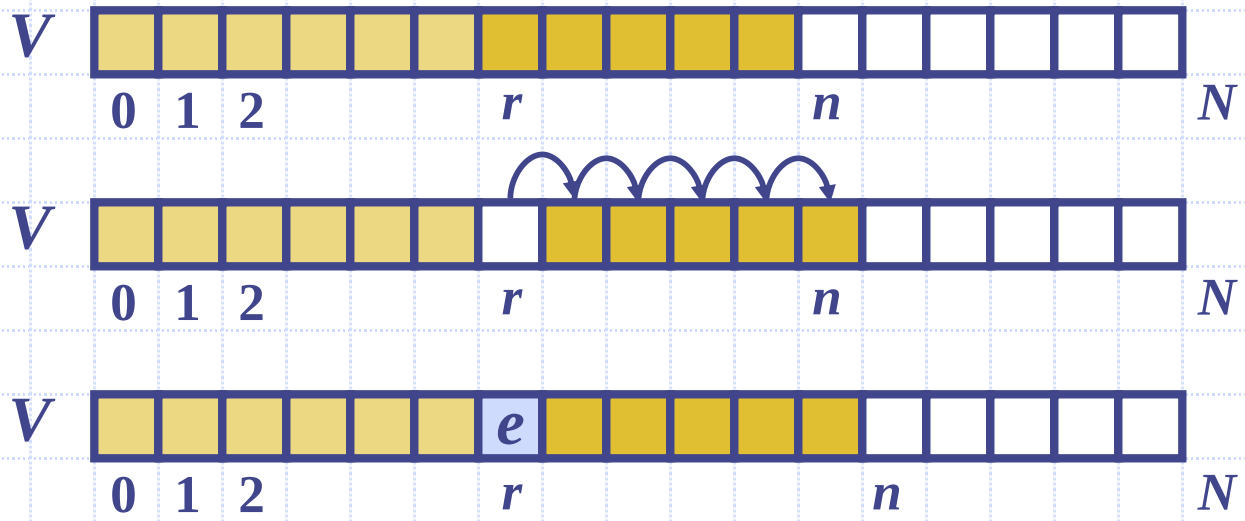
1. **for** $r \leftarrow 0$ **to** $n-1$
   visit($V[r]$) {print, etc}
2. **return**



삽입(insertion)



- 작업 add(r, e)에서는, r 순위로 새 원소 e가 들어갈 빈 자리를 만들기 위해 $V[n-1], \cdots, V[r]$까지의 $n-r$개의 원소들을 **순방향**으로 이동(shift forward)
- 최악의 경우($r=0$), $\mathbf{O}(n)$ 시간 소요



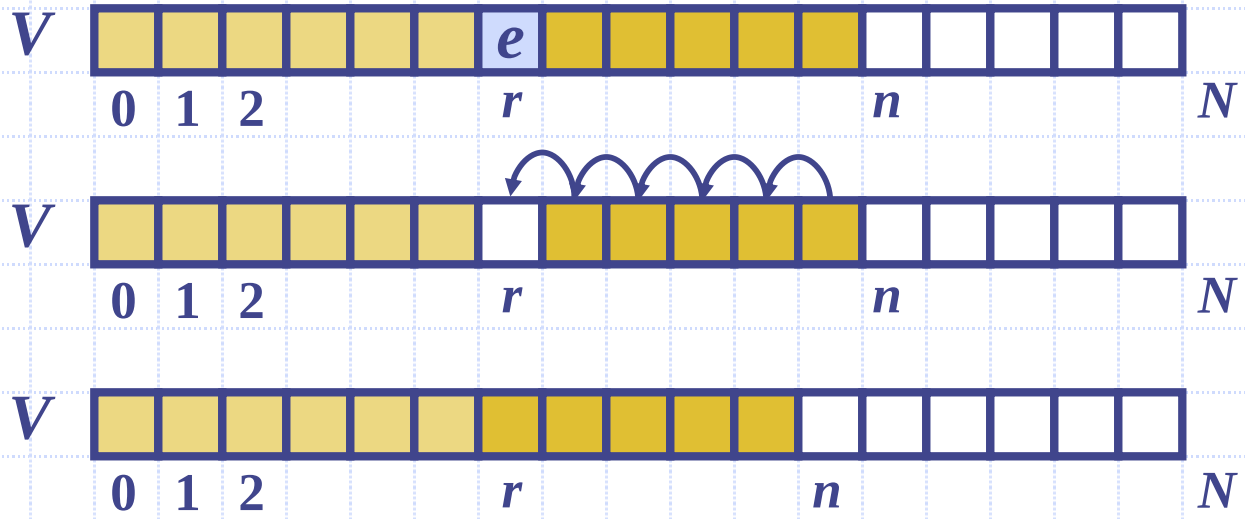
삽입 (conti.)

- 배열의 지정된 순위 ***r***에 원소 ***e***를 삽입

```
Alg add(r, e)
    input array V, integer N, n, rank
          r, element e
    output none

1. if (n = N)
          fullListException()
2. if ((r < 0) || (r > n))
          invalidRankException()
3. for i ← n – 1 downto r
          V[i + 1] ← V[i]
4. V[r] ← e
5. n ← n + 1
6. return
```

삭제(deletion)

- 작업 remove(r)에서는, 삭제된 원소에 의해 생긴 빈 자리를 채우기 위해 $V[r+1], \ldots, V[n-1]$까지의 $n-r-1$개의 원소들을 **역방향**으로 이동(shift backward)
- 최악의 경우($r=0$), $O(n)$ 시간 소요

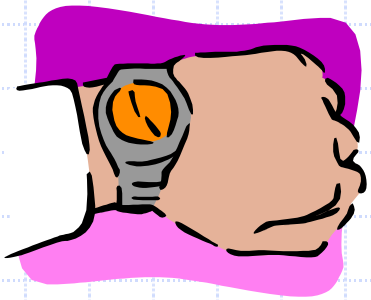


삭제 (conti.)

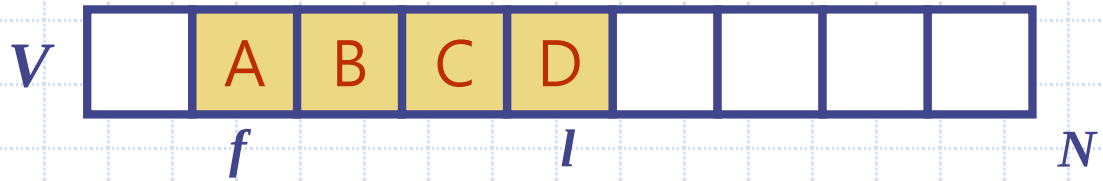
- 배열의 지정된 순위 ***r***의 원소를 삭제하여 반환

```
Alg remove(r)
    input array V, integer N, n, rank
          r
    output element e

1. if ((r < 0) || (r > n – 1))
          invalidRankException()
2. e ← V[r]
3. for i ← r + 1 to n – 1
          V[i – 1] ← V[i]
4. n ← n – 1
5. return e
```

성능(performance)



- **배열**을 이용하여 리스트 ADT를 구현할 경우
  - 데이터구조에 의한 기억장소 사용량: $\mathbf{O}(n)$
  - size, isEmpty, get, set: $\mathbf{O}(1)$
  - add, remove: $\mathbf{O}(n)$
  - addFirst, removeFirst: $\mathbf{O}(n)$
  - addLast, removeLast: $\mathbf{O}(1)$
- 배열을 **원형**으로 이용하면:
  - addFirst, removeFirst: $\mathbf{O}(1)$



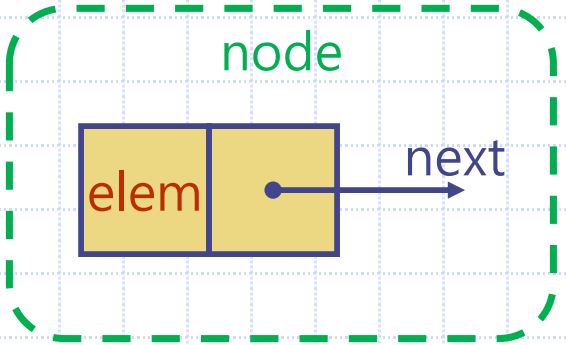
- add 작업에서 배열이 **만원**(full)이면 배열을 동적으로 확장하여 해결 가능(동적 할당)

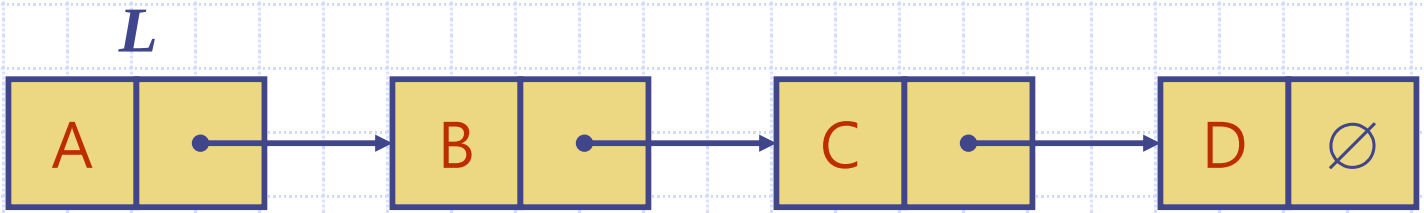
연결리스트를 이용한 구현

- 단일연결리스트 또는,
- 이중연결리스트를 사용

단일연결리스트

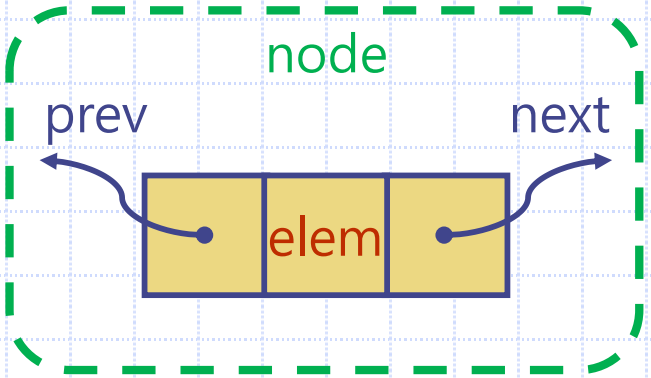
- **단일연결리스트**(singly linked list): 연속 노드로 구성된 구체적인 데이터구조
- 각 노드의 저장 내용
  - **원소**(element) (단순 또는 복잡)
  - 다음 노드를 가리키는 **링크**(link)

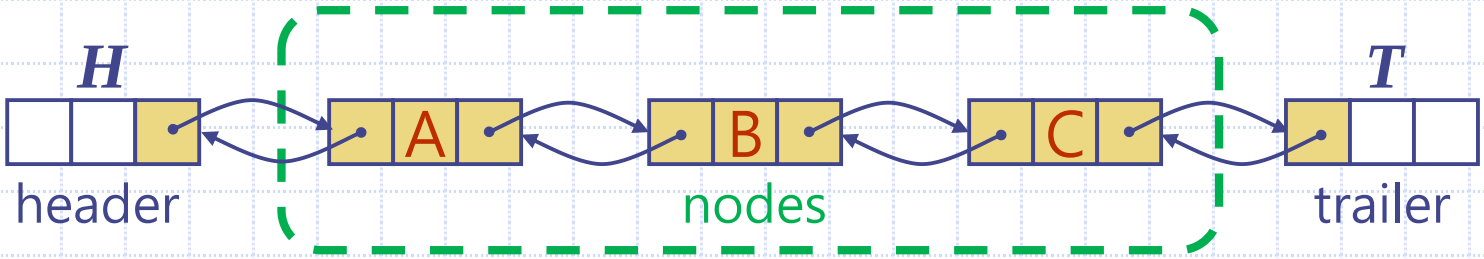




이중연결리스트

- **이중연결리스트**(doubly linked list)를 이용하면 리스트 ADT를 자연스럽게 구현 가능
- 각 노드의 필드
  - **원소**
  - 이전 노드를 가리키는 **링크**
  - 다음 노드를 가리키는 **링크**
- 특별 헤더 및 트레일러 노드





이중연결리스트를 이용한 구현

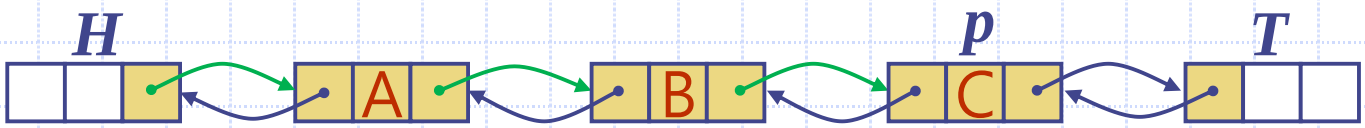
- 작업 get(r) 또는 set(r, e)는 $O(n)$ 시간에 지정된 순위 r의 원소를 반환 또는 저장
- 연결리스트에서 **순위**는 1에서 출발로 전제

Alg *get*(*r*)
  input a doubly linked list with header *H* and trailer *T*, rank *r*
  output element

1. **if** ((*r* < 1) || (*r* > *n*))
   invalidRankException()
2. *p* ← *H*
3. **for** *i* ← 1 **to** *r*
   p ← *p*.next
4. **return** *p*.elem

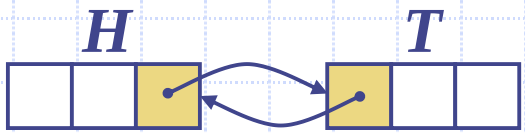
Alg *set*(*r*, *e*)
  input a doubly linked list with header *H* and trailer *T*, rank *r*, element *e*
  output element

1. **if** ((*r* < 1) || (*r* > *n*))
   invalidRankException()
2. *p* ← *H*
3. **for** *i* ← 1 **to** *r*
   p ← *p*.next
4. *p*.elem ← *e*
5. **return** *e*



초기화

- 초기에는 아무 노드도 없다
- **O**(1) 시간 소요



Alg ***initialize***()
input none
output an empty doubly linked list with header ***H*** and trailer ***T***

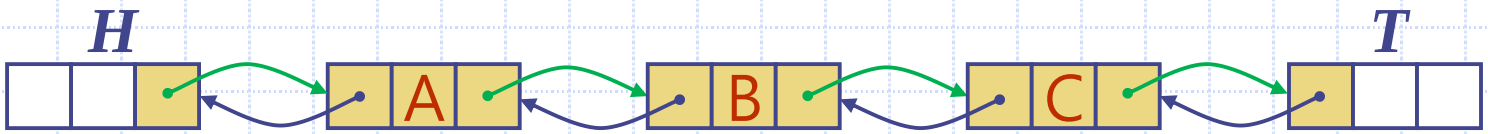
1. ***H*** ← ***getnode***()
2. ***T*** ← ***getnode***()
3. ***H*****.next** ← ***T***
4. ***T*****.prev** ← ***H***
5. ***n*** ← 0 {optional}
6. **return**

순회

- 연결리스트의 모든 원소들을 방문
- **O(*n*)** 시간 소요

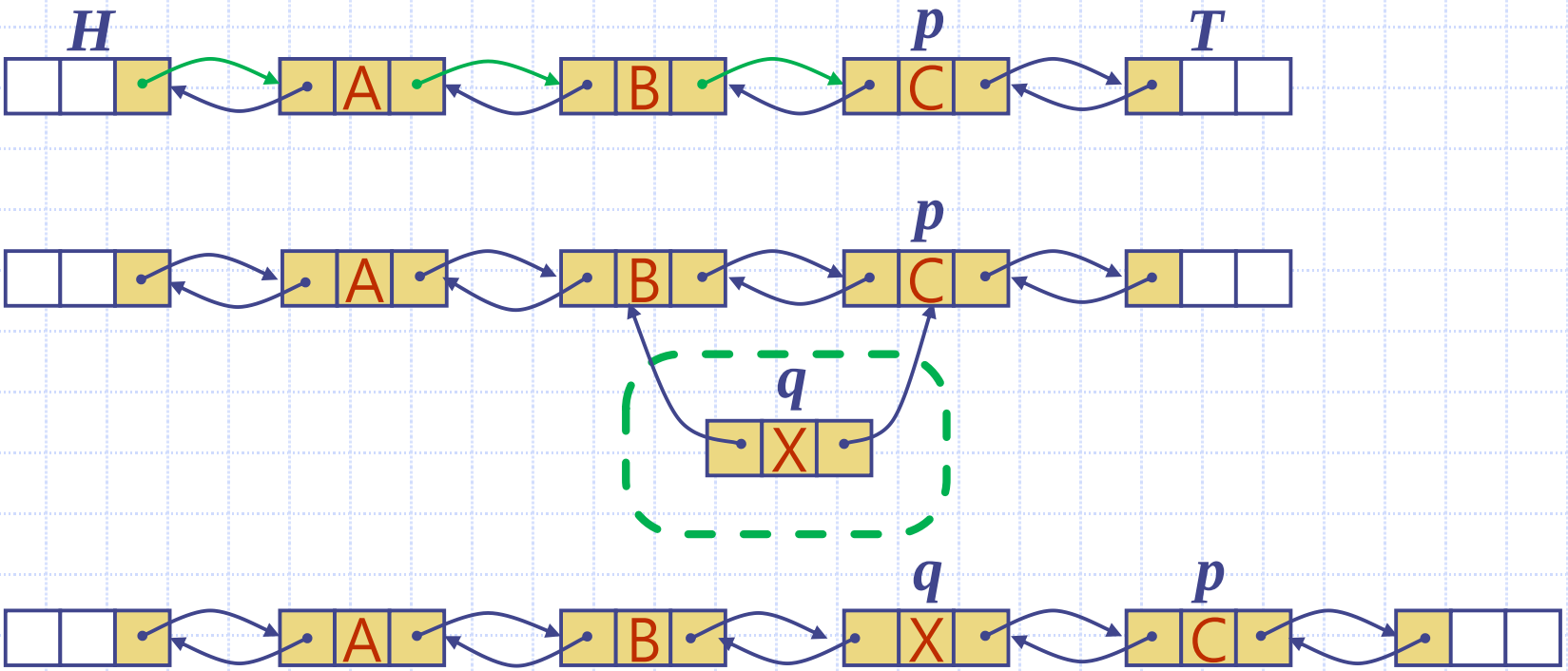
Alg ***traverse***()
input a doubly linked list with header ***H*** and trailer ***T***
output none

1. $p \leftarrow H$**.next**
2. **while** ($p \neq T$)
   visit(***p***.**elem**) {print, etc}
   $p \leftarrow p$**.next**
3. **return**



삽입

- 작업 add(r, X)의 시각화 – 여기서 r = 3



삽입 (conti.)

Alg *add*(*r*, *e*)
  input a doubly linked list with header *H* and trailer *T*, rank *r*, element *e*
  output none

1. **if** ((*r* < 1) || (*r* > *n* + 1))
     invalidRankException()
2. *p* ← *H*
3. **for** *i* ← 1 **to** *r*
     p ← *p*.**next**
4. *addNodeBefore*(*p*, *e*)
5. *n* ← *n* + 1 {optional}
6. **return**

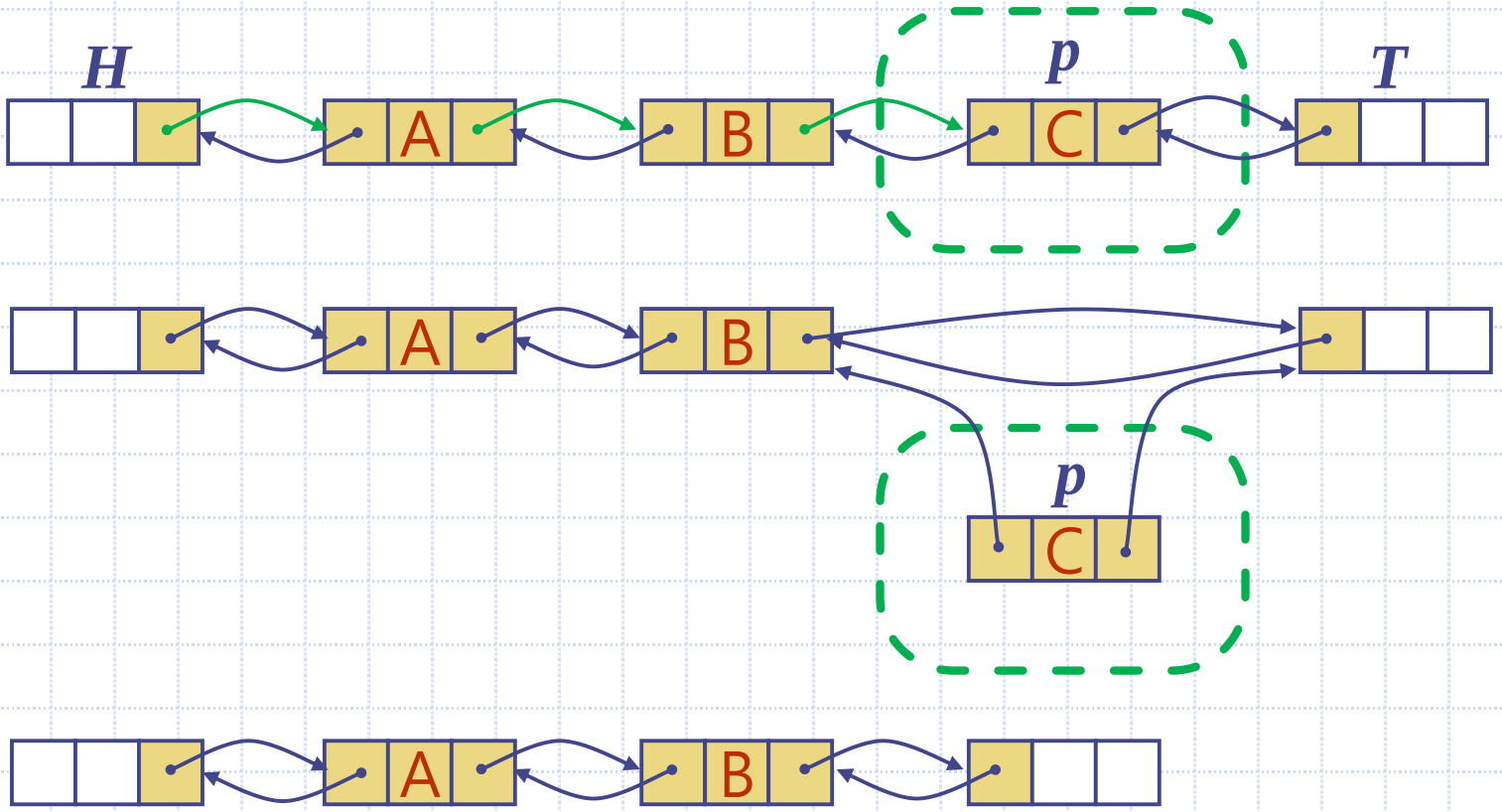
Alg *addNodeBefore*(*p*, *e*)
  input a doubly linked list with header *H* and trailer *T*, node *p*, element *e*
  output none

1. *q* ← *getnode*()
2. *q*.**elem** ← *e*
3. *q*.**prev** ← *p*.**prev**
4. *q*.**next** ← *p*
5. (*p*.**prev**).**next** ← *q*
6. *p*.**prev** ← *q*
7. **return**

- 이중연결리스트의 지정된 순위 *r*에 원소 *e*를 삽입
- add: **O**(*n*) 시간 소요

삭제

- remove(r)의 시각화 – 여기서 r = 3



삭제 (conti.)

Alg *remove*(*r*)
  input a doubly linked list with header *H* and trailer *T*, rank *r*
  output element

1. **if** ((*r* < 1) || (*r* > *n*))
     invalidRankException()
2. *p* ← *H*
3. **for** *i* ← 1 **to** *r*
     p ← *p*.**next**
4. *e* ← *removeNode*(*p*)
5. *n* ← *n* – 1 {optional}
6. **return** *e*

Alg *removeNode*(*p*)
  input a doubly linked list with header *H* and trailer *T*, node *p*
  output element

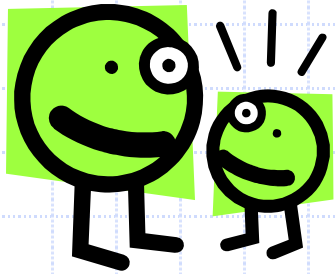
1. *e* ← *p*.**elem**
2. (*p*.**prev**).**next** ← *p*.**next**
3. (*p*.**next**).**prev** ← *p*.**prev**
4. *putnode*(*p*) {reuse}
5. **return** *e*

- 이중연결리스트의 지정된 순위 *r*의 노드를 삭제하고 원소를 반환
- remove: **O**(*n*) 시간 소요

성능

- **이중연결리스트**를 이용한 리스트 ADT를 구현할 경우
  - 연결리스트의 각 원소에 사용되는 기억장소: $\mathbf{O}(1)$
  - n개의 원소로 구성된 연결리스트에 의해 사용되는 기억장소: $\mathbf{O}(n)$
  - size, isEmpty: $\mathbf{O}(1)$
  - get, set, add, remove: $\mathbf{O}(n)$
  - addFirst, addLast, removeFirst, removeLast: $\mathbf{O}(1)$

성능 요약

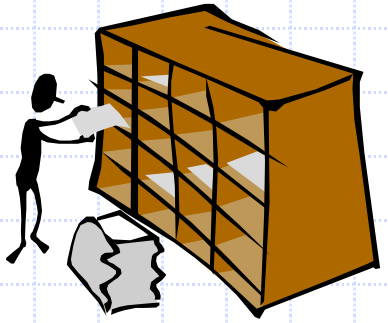


작업	배열	연결리스트
size, isEmpty	1	1
get, set	1	*n*
add, remove	*n*	*n*
addFirst, removeFirst	*n*	1
addLast, removeLast	1	1

리스트 확장: 그룹과 공유

- 리스트 ADT를 확장하여 설계 가능한 고차원의 개념들을 연구한다
- 대상 개념들
  - 그룹(grouping)
  - 공유(sharing)

그룹



- **개념:** 데이터원소들이 각각 상이한 **그룹**(즉, 카테고리)에 속한다
- **전제:** 각 그룹의 크기는 다양

- **예**
  - 쇼핑몰의 상품들
    - Maker X: x1
    - Maker Y: none
    - Maker Z: z1, z2
  - 대학의 강좌들
    - Prof. Kook: DS
    - Prof. Park: (no lecture)
    - Prof. Shin: DB, MM
  - 다항식의 항들
    - Exp 1: $3x^4$
    - Exp 3: $5x^3$, -4

설계 방안

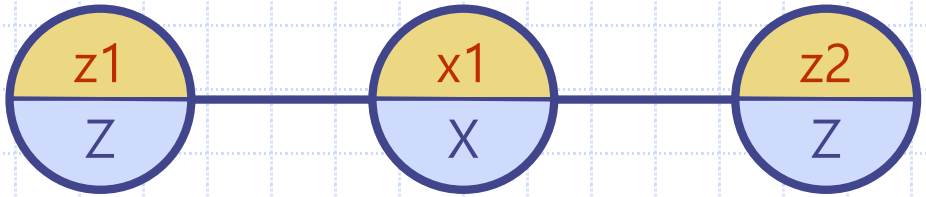
A. 레코드의 리스트 사용
   1. 배열을 이용한 구현
   2. 연결리스트를 이용한 구현

B. 부리스트(sublist)들의 리스트 사용
   1. 2D 배열을 이용한 구현
   2. 연결리스트의 배열을 이용한 구현

설계 방안 A: 레코드의 리스트 사용

- 그룹을 표현하기 위해, **elem** 및 **group** 필드로 구성된 **레코드**의 **리스트**를 사용
- **장점:** 단순
- **단점:** 특정그룹에 관한 작업을 위해서는 전체 레코드를 순회
  - $O(n)$ 시간 소요. 단, n은 총 레코드 개수

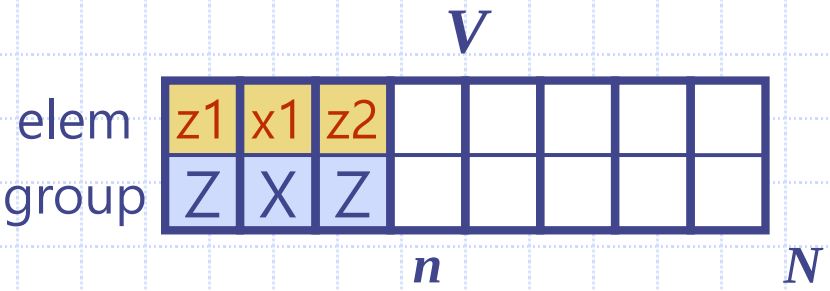
예: 쇼핑몰의 상품들



설계 방안 A의 구현 1: 배열 이용

- 리스트를 **elem** 및 **group** 필드로 구성된 **레코드**의 **1D 배열**을 이용하여 구현
- **장점:** 기억장소 낭비 없음

예: 쇼핑몰의 상품들

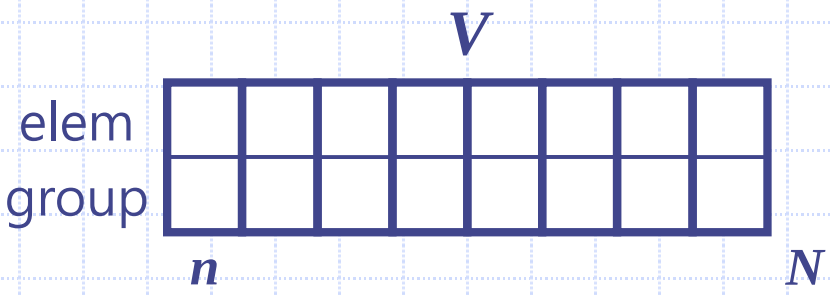


초기화

- 초기에는, 각 그룹에 아무 레코드도 없다

Alg ***initGroup***()
input array ***V***, integer ***N***, ***n***
output an empty array ***V*** of size ***n***

1. ***n*** ← 0
2. **return**



순회

- 지정된 그룹의 모든 멤버들을 방문

```
Alg traverseGroup(g)
    input array V, integer N, n,
          group g
    output none

1. for i ← 0 to n – 1
        if (V[i].group = g)
            visit(V[i].elem)
2. return
```

삭제

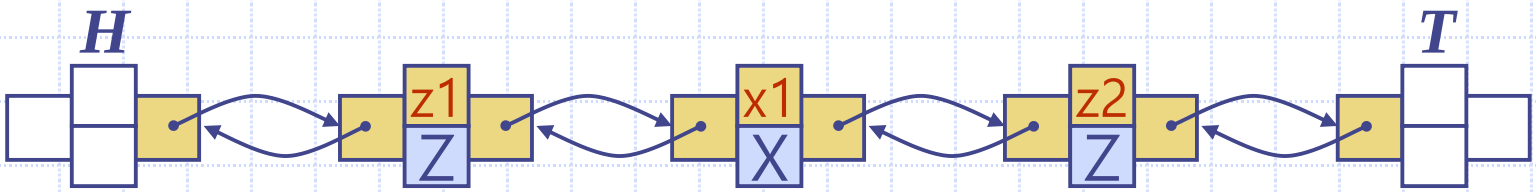
- 지정된 그룹의 모든 멤버를 삭제

```
Alg removeGroup(g)
    input array V, integer N, n,
          group g
    output none

1. i ← 0
2. while (i < n)
        if (V[i].group = g)
            remove(i)
        else
            i ← i + 1
3. return
```

설계 방안 A의 구현 2: 연결리스트 이용

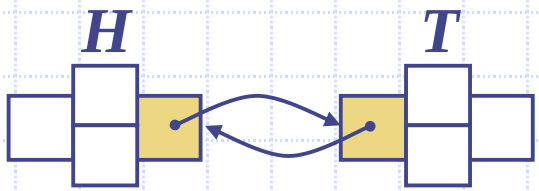
- 리스트를 **elem** 및 **group** 필드로 구성된 **레코드** 노드의 **이중연결리스트**를 이용하여 구현 가능
- **장점:** 기억장소 사용 최소화

예: 쇼핑몰의 상품들



초기화

- 초기에는 각 그룹에 아무 노드도 없다



Alg ***initGroup***()
input none
output an empty doubly linked list with header ***H*** and trailer ***T***

1. ***H*** ← ***getnode***()
2. ***T*** ← ***getnode***()
3. ***H*.next** ← ***T***
4. ***T*.prev** ← ***H***
5. ***n*** ← 0 {optional}
6. **return**

순회

- 지정된 그룹의 모든 멤버들을 방문

Alg ***traverseGroup***(***g***)
  input a doubly linked list with header ***H*** and trailer ***T***, group ***g***
  output none

1. ***p*** ← ***H*****.next**
2. **while** (***p*** ≠ ***T***)
     if (***p*****.group** = ***g***)
       visit(***p*****.elem**)
     p ← ***p*****.next**
3. **return**

삭제

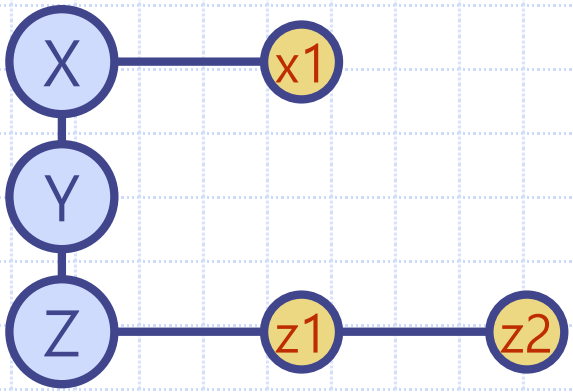
- 지정된 그룹의 모든 멤버들을 삭제

```
Alg removeGroup(g)
    input a doubly linked list with
          header H and trailer T, group
          g
    output none

1. p ← H.next
2. while (p ≠ T)
        pnext ← p.next   {save next}
        if (p.group = g)
            removeNode(p)
            n ← n – 1
        p ← pnext        {restore next}
3. return
```

설계 방안 B: 부리스트들의 리스트 사용

- 그룹을 표현하기 위해, 그룹의 **리스트**를 사용하며, 각 그룹은 다시 원소들의 **부리스트**로 구성
- **장점:** 특정그룹 관련 작업 격리 처리 가능

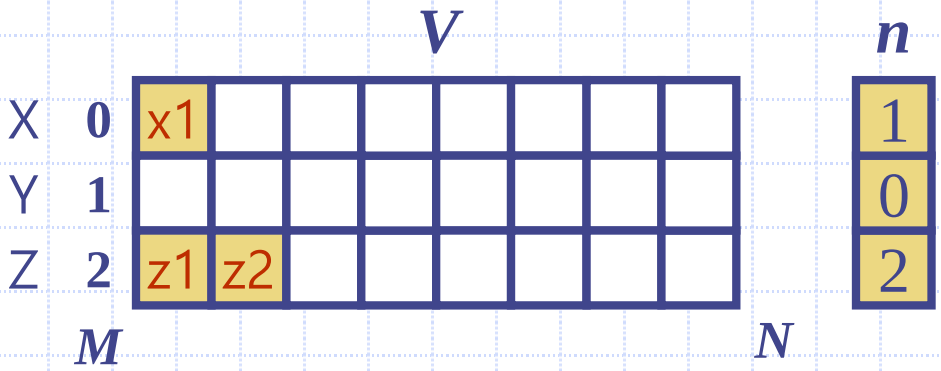
예: 쇼핑몰의 상품들



설계 방안 B의 구현 1: 2D 배열 이용

- $M \times N$ **배열**을 사용하여, 리스트는 각 그룹을 나타내는 행을, 부리스트는 각 행의 원소들을 이용하여 구현 – 단, M은 그룹의 개수
- **단점:** 열의 크기 N이 최대 그룹의 크기를 커버해야 하므로 기억장소 낭비의 우려

예: 쇼핑몰의 상품들

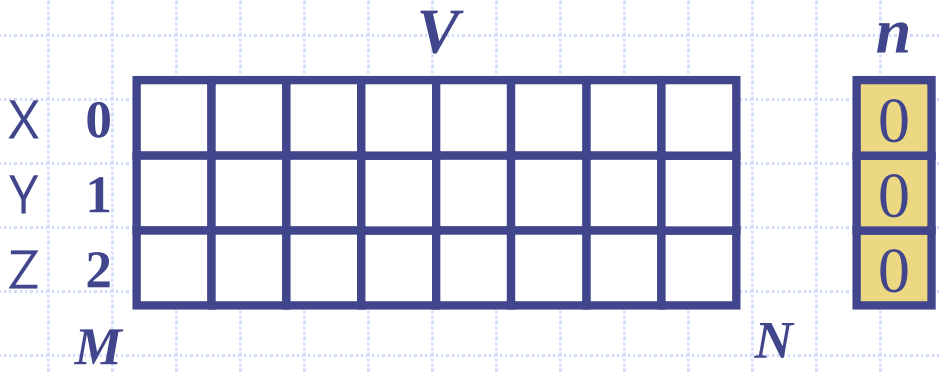


초기화

- 초기에는 각 그룹에 아무 멤버도 없다

Alg ***initGroup***()
input array ***V***, ***n***, integer ***N***, ***M***
output array ***n*** of size ***M***

1. **for** ***i*** ← 0 **to** ***M*** – 1
   n[***i***] ← 0
2. **return**



순회

- 지정된 그룹의 모든 멤버들을 방문

```
Alg traverseGroup(g)
    input array V, n, integer N,
          group g
    output none

1. for j ← 0 to n[g] – 1
        visit(V[g, j])
2. return
```

삭제

- 지정된 그룹의 모든 멤버들을 삭제

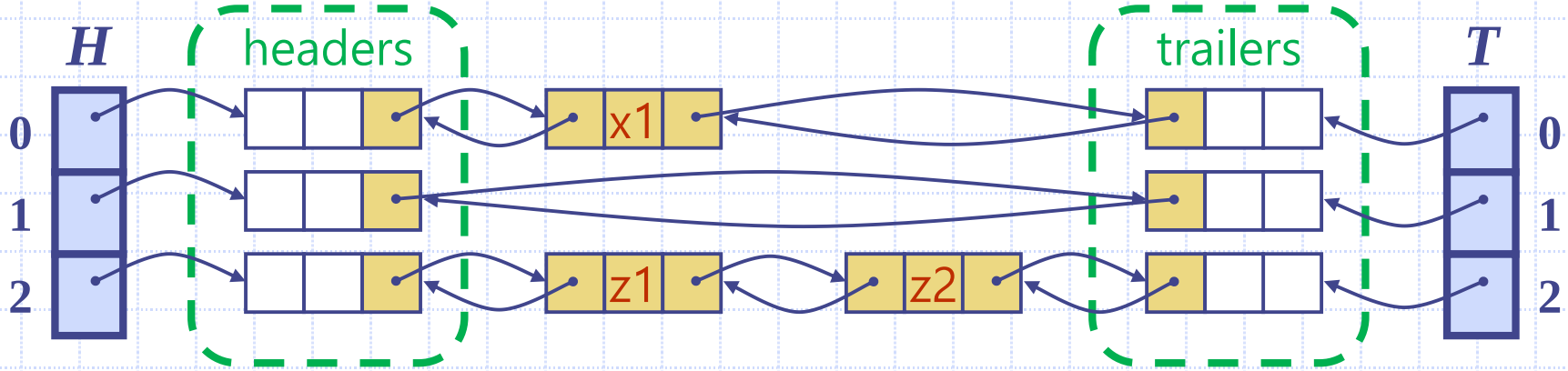
Alg ***removeGroup***(***g***)
input array ***V***, ***n***, integer ***N***,
group ***g***
output none

1. ***n***[***g***] ← 0
2. **return**

설계 방안 B의 구현 2: 연결리스트의 배열 이용

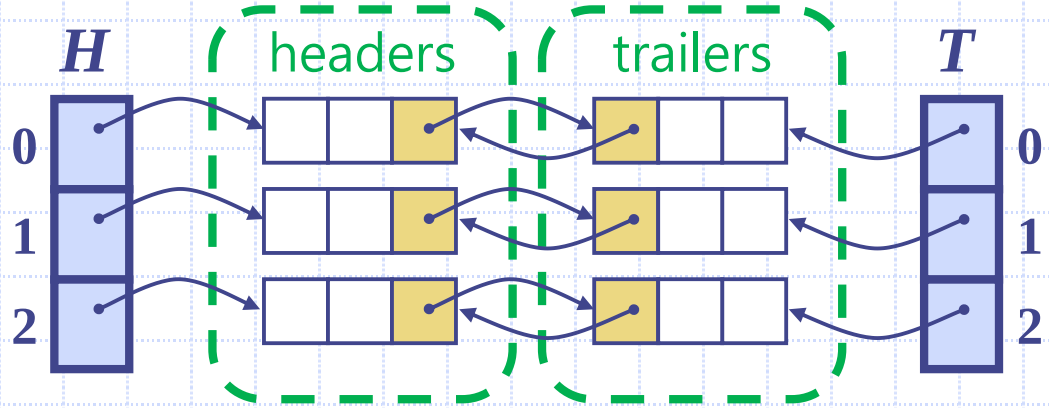
- 리스트는 헤더 및 트레일러 주소를 저장하기 위한 두 개의 **1D 배열**로, 부리스트는 각 그룹에 대한 **이중연결리스트**로 구현
  - 두 개의 1D 배열을 한 개의 2D 배열로 통합 가능
- **장점:** 기억장소 사용 최소화

예: 쇼핑몰의 상품들



초기화

- 초기에는 각 그룹에 아무 노드도 없다



초기화 (conti.)

Alg ***initGroup***()
input array ***H***, ***T***, integer ***M***
output array ***H***, ***T*** of pointers,
each to headers and trailers of
empty doubly linked lists

1. **for** ***i*** ← 0 **to** ***M*** – 1
h ← ***getnode***()
t ← ***getnode***()
h*.next** ← ***t
t*.prev** ← ***h
H[***i***] ← ***h***
T[***i***] ← ***t***
2. **return**

순회

- 지정된 그룹의 모든 멤버들을 방문

```
Alg traverseGroup(g)
    input array H, T, group g
    output none

1. p ← H[g].next
2. while (p ≠ T[g])
        visit(p.elem)
        p ← p.next
3. return
```

삽입

- 지정된 그룹의 지정된 위치(여기서는 맨 앞)에 원소를 삽입

Alg ***addGroupFirst***(***g***, ***e***)
    input array ***H***, ***T***, group ***g***,
        element ***e***
    output none

1. ***H***[***g***].***addFirst***(***e***)
2. **return**

삭제

- 지정된 그룹의 모든 멤버들을 삭제

Alg ***removeGroup***(***g***)
input array ***H***, ***T***, group ***g***
output none

1. ***removeAll***(***H***[***g***], ***T***[***g***]){exercise}
2. **return**

리스트 확장: 공유

- **문제 상황:** 데이터 원소(element)들이 상이한 그룹(group)에 의해 **공유**(share)됨 (**예:** auction)
- **전제:** 각 관련 그룹에게 공유 데이터원소를 복제하는 것은 시간과 기억장소가 낭비되므로 허용하지 않는다
- **예**
  - 쇼핑몰의 상품들
    - Buyer A: x1, y1
    - Buyer B: z1
    - Buyer C: y1, z1, z2
  - 대학 강좌들
    - Student A: DS, OS
    - Student B: DB
    - Student C: OS, DB, MM
  - 인터넷 블로그들
    - Blogger A: a, b
    - Blogger B: c
    - Blogger C: b, c, d

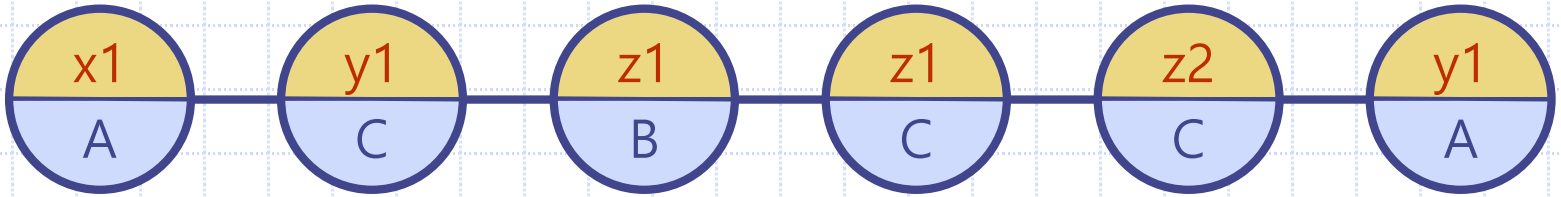
설계 방안: 공유

A. 레코드의 리스트 사용
   1. 배열을 이용한 구현
   2. 연결리스트를 이용한 구현
B. 포인터의 리스트 사용
   1. 배열을 이용한 구현
   2. 연결리스트를 이용한 구현
C. 다중리스트 사용
   1. 2D 배열 사용
   2. 다중 연결리스트를 이용한 구현

설계 방안 A: 레코드의 리스트 사용

- 앞서의 그룹을 표현하는 설계 방안 A와 동일
- 공유를 표현하기 위해, **elem** 및 **group** 필드로 구성된 **레코드**의 **리스트**를 사용
- **단점:** 특정원소 및 특정그룹 관련 작업 모두에 전체 레코드 순회 필요
  - 각각 $O(n)$ 시간 소요, 단, n은 총 레코드 개수

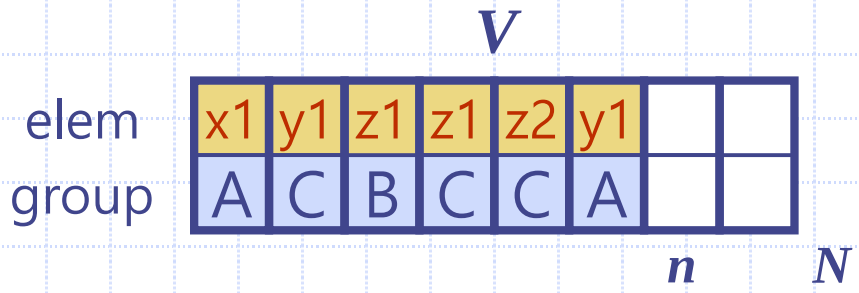
예: 쇼핑몰의 상품들



설계 방안 A의 구현 1: 배열 이용

- 리스트를 **elem** 및 **group** 필드로 구성된 **레코드**의 **1D 배열**로 구현
- **장점:** 기억장소 낭비 없음

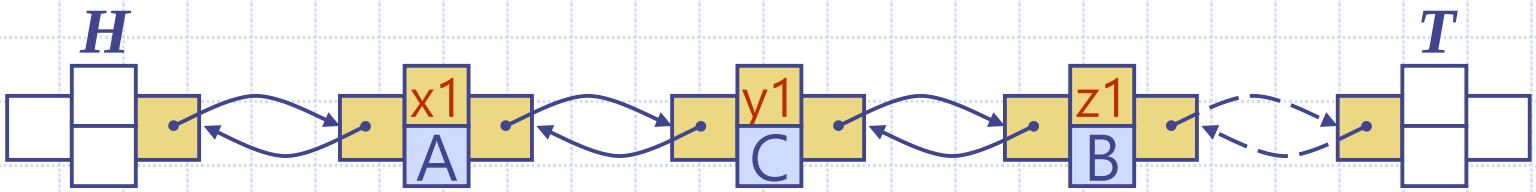
예: 쇼핑몰의 상품들



설계 방안 A의 구현 2: 연결리스트 이용

- 리스트를 **elem** 및 **group** 필드로 구성된 **레코드** 노드의 **이중연결리스트**로 구현
- **장점:** 기억장소 낭비 최소화

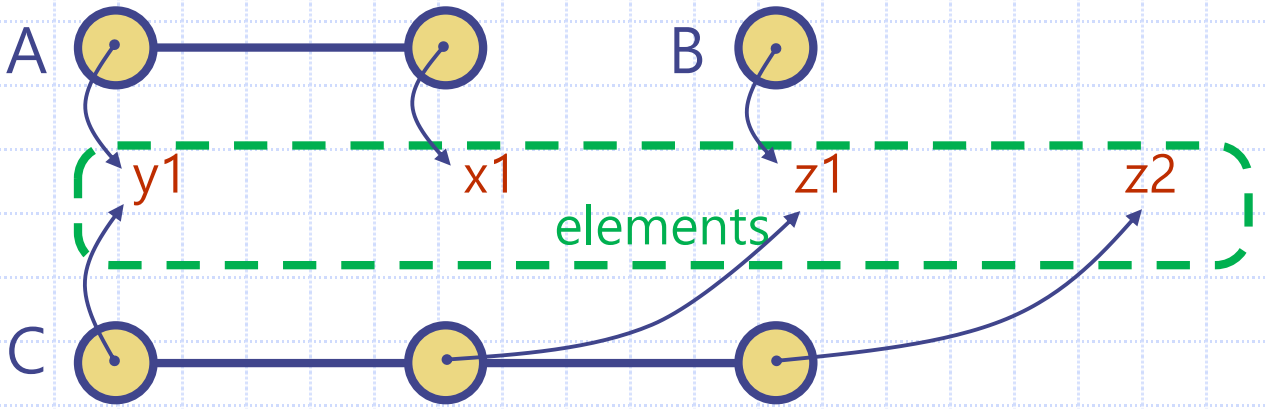
예: 쇼핑몰의 상품들



설계 방안 B: 포인터의 리스트 사용

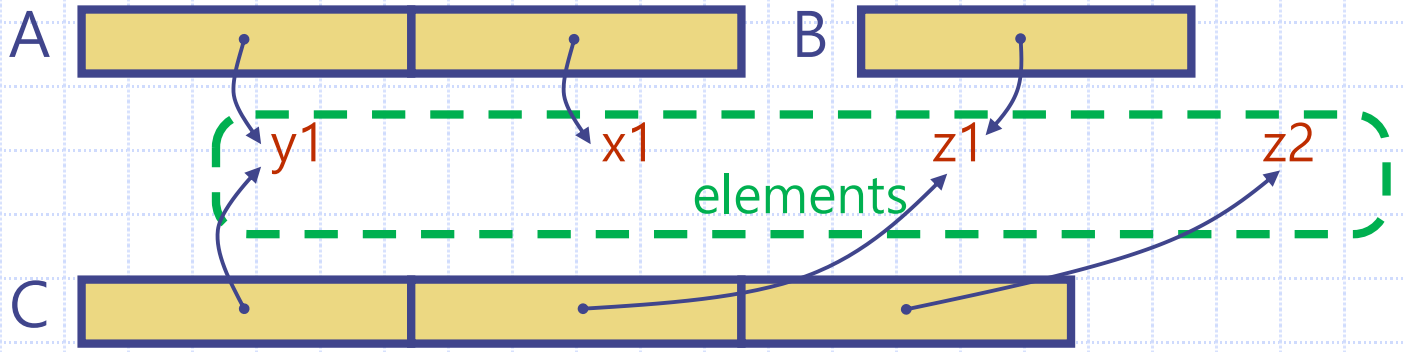
- 공유를 표현하기 위해, **원소**들을 별도의 메모리에 저장하고 이들에 대한 참조를 **포인터**를 통해 수행
- **장점:** 단순, 기억장소 사용 최소화
- **단점:** 특정원소 관련작업의 격리 처리 불가

예: 쇼핑몰의 상품들

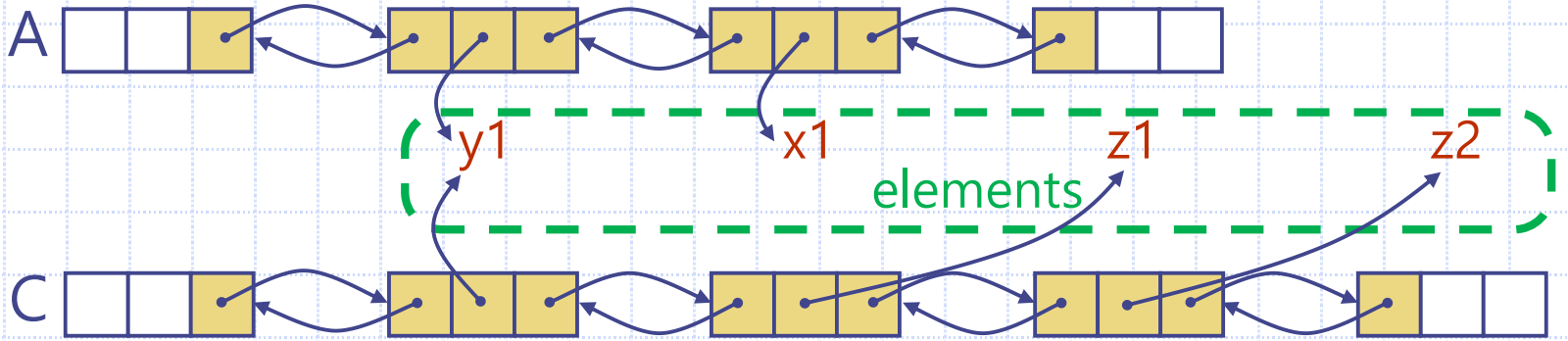


설계 방안 B의 두 가지 구현

1. **배열** 이용



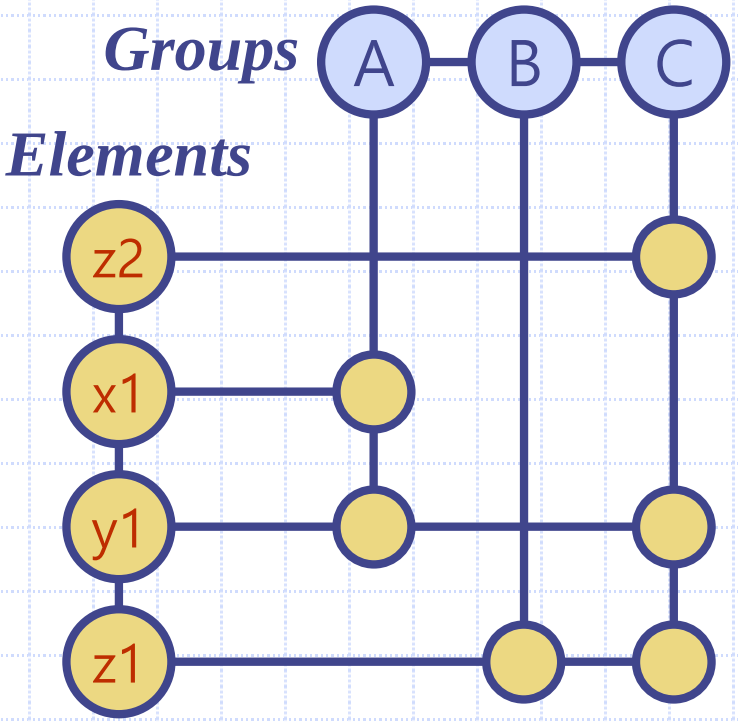
2. **연결리스트** 이용 (B는 생략됨)



설계 방안 C: 다중리스트 사용

- 공유를 표현하기 위해, **원소**들의 리스트와 **그룹**들의 리스트가 상호 교차하는 형태의 다중리스트(multilist)를 사용
- 교차점 서브리스트는 (원소, 그룹) 관련성 여부를 표현
- **장점:** 특정원소 및 특정그룹 관련 작업 모두 격리 처리 가능

예: 쇼핑몰의 상품들



설계 방안 C의 구현 1: 2D 배열 이용

- 행과 열이 각각 원소와 그룹을 나타내는 **2D 논리**(boolean) **배열**을 이용
- **단점:** 원소-그룹 간 관계가 희소한 경우, 기억장소 낭비

예: 쇼핑몰의 상품들

V

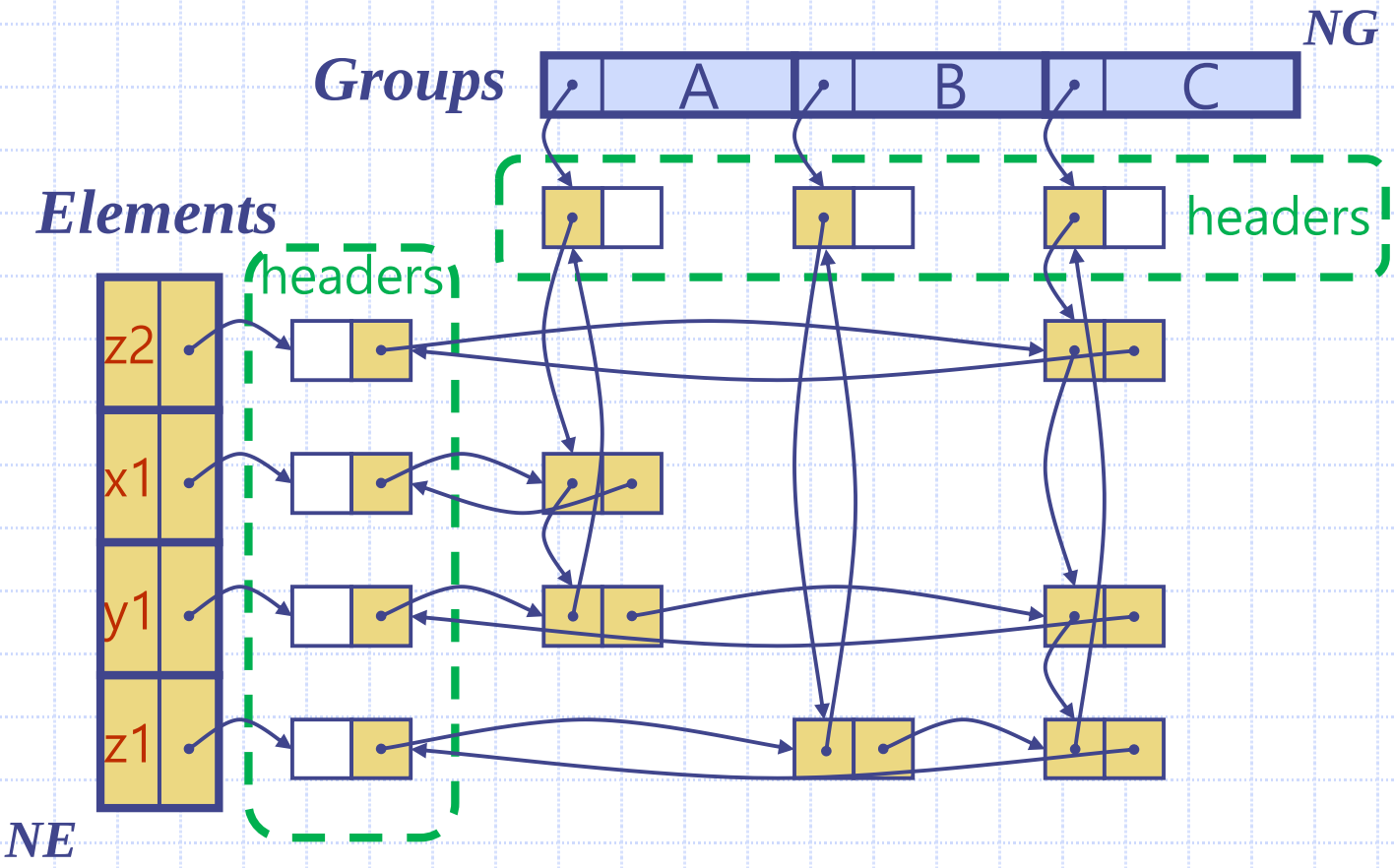
	A	B	C
z2	0	0	1
x1	1	0	0
y1	1	0	1
z1	0	1	1

설계 방안 C의 구현 2: 다중연결리스트 이용

- 다음과 같이 **다중연결리스트**(multilinkedlist) 구현
  - 두 개의 **배열**을 이용하여 원소 및 그룹 리스트를 각각 구현
  - 상호교차하는 **원형 헤더 연결리스트**들을 이용하여 (원소, 그룹) 쌍의 부리스트들을 구현
    - 헤더 외에 트레일러도 이용하거나, 또는 이중 연결리스트 이용 가능
- **장점:** 기억장소 낭비 최소화
- 교차점 노드의 원소 및 그룹 정보는 해당 부리스트의 링크를 추적하여 헤더에서 구하도록 구현 가능
  - 또는, 각 교차점 노드에 원소 및 그룹 헤더로 직행하는 포인터들을 추가

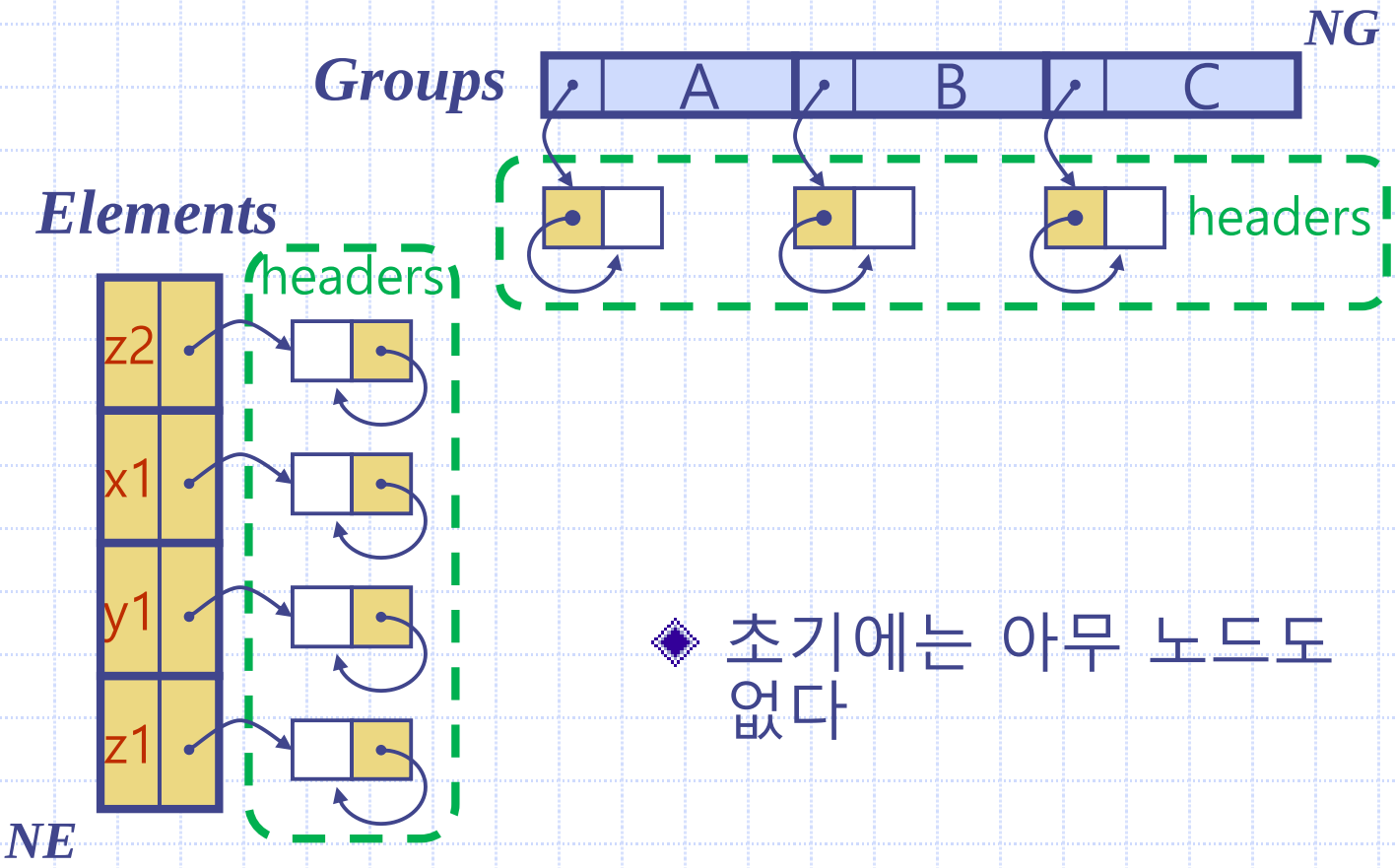
다중연결리스트 이용 (conti.)

예: 쇼핑몰의 상품들



초기화

예: 쇼핑몰의 상품들



- 초기에는 아무 노드도 없다

초기화 (conti.)

```
Alg initShare()
    input array Elements, Groups, integer
          NE, NG
    output an empty multilinkedlist

1. for i ← 0 to NE – 1
          H ← getnode()
          H.nextgroup ← H
          Elements[i].header ← H     {null list}
2. for i ← 0 to NG – 1
          H ← getnode()
          H.nextelement ← H
          Groups[i].header ← H       {null list}
3. return
```

원소 순회

- 지정된 그룹과 관련된 모든 원소들을 방문
- **예:** 구매자 *C*가 구입한 상품들을 모두 열거

Alg ***traverseShareElements***(*g*)
  input array ***Groups***, group ***g***
  output none

1. ***H*** ← ***Groups***[*g*].**header**
2. ***p*** ← ***H***.**nextelement**
3. **while** (***p*** ≠ ***H***)
   visit(***p***)
   p ← ***p***.**nextelement**
4. **return**

그룹 순회

- 지정된 원소와 관련된 모든 그룹들을 방문
- **예:** 상품 *z1*을 구입한 구매자들을 모두 열거

Alg ***traverseShareGroups*(*e*)**
input array ***Elements***, element ***e***
output none

1. ***H*** ← ***Elements***[***e***].**header**
2. ***p*** ← ***H***.**nextgroup**
3. **while** (***p*** ≠ ***H***)
   ***visit*(*p*)**
   p ← ***p***.**nextgroup**
4. **return**

삽입

- (원소, 그룹) 쌍을 지정된 위치(여기서는 맨 앞)에 삽입
- **예:** 매매기록 (*z1*, *C*)를 추가

Alg ***addShare*(*e*, *g*)**
input array ***Elements***, ***Groups***,
element ***e***, group ***g***
output none

1. ***p*** ← ***getnode***()
2. ***HG*** ← ***Groups***[***g***].**header**
3. ***p*.nextelement** ← ***HG*.nextelement**
4. ***HG*.nextelement** ← ***p***
5. ***HE*** ← ***Elements***[***e***].**header**
6. ***p*.nextgroup** ← ***HE*.nextgroup**
7. ***HE*.nextgroup** ← ***p***
8. **return**

삭제

- **단일연결리스트**를 이용하여 구현할 경우, (e, g) 노드 한 개, 또는 특정 원소나 특정 그룹의 모든 (e, g) 노드를 삭제하는데 많은 시간 소요 – **이유:** 삭제 작업에 이전 노드를 참조하는 것이 필요하기 때문
- 삭제 시간을 단축하기 위해 헤더와 트레일러가 있는 **이중연결리스트**를 이용하여 구현하는 것도 가능

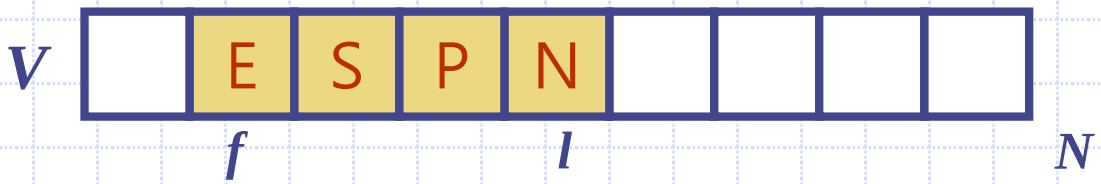
응용문제: 원형배열

- 리스트 ADT를 배열에 기초하여 구현하기 위한 데이터구조를 설계하고 관련 메쏘드를 작성하는 문제
- **주의**
  - 삽입과 삭제 메쏘드들은 가능한 시간 효율이 좋도록 작성되어야 하며, 특히 첫 순위와 끝 순위에서의 삽입과 삭제는 모두 $\mathbf{O}(1)$ 시간에 실행되어야 한다
  - 메쏘드 get은 상수시간에 실행되어야 한다

A. 설계한 데이터구조를 간략히 설명하라
B. 다음 메쏘드들을 의사코드로 작성하라
   - size(), get(r), set(r, e)
   - add(r, e)
   - remove(r)

답

- **원형배열** 사용
- 이를 위해, 두 개의 첨자 변수 f 와 l을 사용하여 각각 첫 순위(rank = 0) 및 끝 순위(rank = $n - 1$)의 원소를 가리키도록 정의 – 여기서 n은 원소의 개수
- 알고리즘에서는 "나머지 N 계산(modular N arithmetic)"을 사용 – 여기서 N은 배열의 크기



답 (conti.)

Alg ***size***()
1. **return** $(N - f + l + 1) \% N$

Alg ***get***(r)
1. **if** $((r < 0) \| (r \geq size()))$
   invalidRankException()
2. **return** $A[(f + r) \% N]$

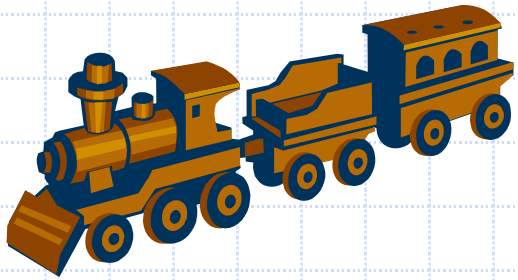
Alg ***set***(r, e)
1. **if** $((r < 0) \| (r \geq size()))$
   invalidRankException()
2. $A[(f + r) \% N] \leftarrow e$
3. **return** e

답 (conti.)

```
Alg add(r, e)
1. n ← size()
2. if (n = N – 1)                {reserve 1 cell}
       fullListException()
3. if ((r < 0) || (r > n))
       invalidRankException()
4. if (r < n/2)
       for i ← 0 to r – 1
           A[(N + f + i – 1) % N] ← A[(f + i) % N]
       A[(N + f + r – 1) % N] ← e
       f ← (N + f – 1) % N
   else
       for i ← n – 1 downto r
           A[(f + i + 1) % N] ← A[(f + i) % N]
       A[(f + r) % N] ← e
       l ← (l + 1) % N
5. return
```

```
Alg remove(r)
1. n ← size()
2. if (n = 0)
       emptyListException()
3. if ((r < 0) || (r ≥ n))
       invalidRankException()
4. e ← A[(f + r) % N]
5. if (r < n/2)
       for i ← r – 1 downto 0
           A[(f + i + 1) % N] ← A[(f + i) % N]
       f ← (f + 1) % N
   else
       for i ← r + 1 to n – 1
           A[(f + i – 1) % N] ← A[(f + i) % N]
       l ← (N + l – 1) % N
6. return e
```

응용문제: 다항식



- 하나의 **다항식**(polynomial)을 하나의 **헤더 단일연결리스트**로 표현하는 방식을 이용하여 여러 개의 다항식을 저장하는 데이터구조를 설계하라
- 다음 세 개의 다항식을 사용하여 위의 설계를 도식적으로 나타내 보여라
  - $a = 3x^4 + 8x$
  - $b = 11x^3 - 8x + 4$
  - $c = -6x^2$

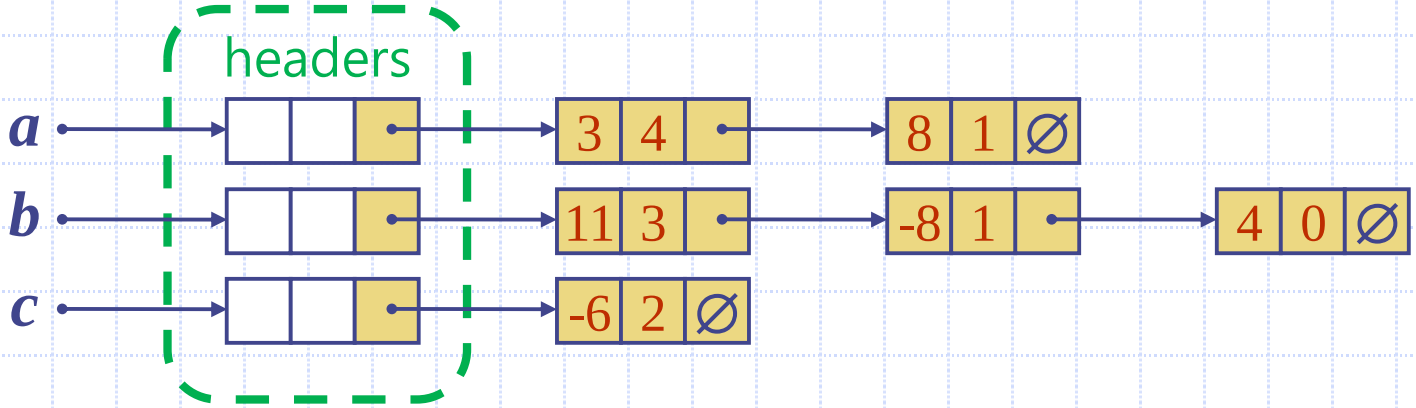
응용문제: 다항식 (conti.)

- 두 개의 다항식 x, y에 대한 **덧셈** 및 **뺄셈**을 수행하여 그 결과를 새로운 헤더 단일연결리스트에 저장, 반환하는 알고리즘 addPoly(x, y) 및 subPoly(x, y)를 **의사코드**로 작성하라
  - **예:** addPoly(a, b)는 $3x^4 + 11x^3 + 4$ 를 반환
  - **예:** subPoly(c, b)는 $-11x^3 - 6x^2 + 8x - 4$ 를 반환
- **힌트:** addPoly 및 subPoly 모두가 호출하는 부 알고리즘 appendTerm(k, c, e)을 먼저 **의사코드**로 작성하라
  - appendTerm(k, c, e): 기존 다항식의 마지막 항을 표현하는 노드 k에 계수(coefficient) c와 차수(exponent) e로 이루어진 새 항을 추가

해결

- 각 다항식에 대해 **헤더 단일연결리스트**를 사용
- 다항식의 각 항을 표현하는 각 노드는 다음 두 개의 필드를 저장:
  - coef: 항의 계수
  - exp: 항의 차수

polynomials



해결 (conti.)

Alg ***appendTerm***(***k***, ***c***, ***e***)
input last term ***k*** of a polynomial expression, coefficient ***c***, exponent ***e***
output cx^e appended to ***k***

1. ***t*** ← ***getnode***()
2. ***t*.coef**, ***t*.exp**, ***t*.next** ← ***c***, ***e***, ∅
3. ***k*.next** ← ***t***
4. ***k*** ← ***t*** {update ***k*** to ***t***}
5. **return**

해결 (conti.)

```
Alg addPoly(x, y)
    input polynomial expression x, y
    output x + y

1. result ← getnode()          {new header}
2. result.next ← ∅             {may be null}
3. i, j ← x.next, y.next       {skip headers}
4. k ← result
5. while ((i ≠ ∅) & (j ≠ ∅))
        if (i.exp > j.exp)
            appendTerm(k, i.coef, i.exp)
            i ← i.next
        elseif (i.exp < j.exp)
            appendTerm(k, j.coef, j.exp)
            j ← j.next
        else
            sum ← i.coef + j.coef
            if (sum ≠ 0)
                appendTerm
                    (k, sum, i.exp)
            i, j ← i.next, j.next
6. while (i ≠ ∅)
        appendTerm(k, i.coef, i.exp)
        i ← i.next
7. while (j ≠ ∅)
        appendTerm(k, j.coef, j.exp)
        j ← j.next
8. return result
```

해결 (conti.)

```
Alg subPoly(x, y)
    input polynomial expression x, y
    output x – y

1. result ← getnode()          {new header}
2. result.next ← ∅             {may be null}
3. i, j ← x.next, y.next       {skip headers}
4. k ← result
5. while ((i ≠ ∅) & (j ≠ ∅))
        if (i.exp > j.exp)
            appendTerm(k, i.coef, i.exp)
            i ← i.next
        elseif (i.exp < j.exp)
            appendTerm(k, –j.coef, j.exp)
            j ← j.next
        else
            diff ← i.coef – j.coef
            if (diff ≠ 0)
                appendTerm
                    (k, diff, i.exp)
            i, j ← i.next, j.next
6. while (i ≠ ∅)
        appendTerm(k, i.coef, i.exp)
        i ← i.next
7. while (j ≠ ∅)
        appendTerm(k, –j.coef, j.exp)
        j ← j.next
8. return result
```

응용문제: 생일케이크

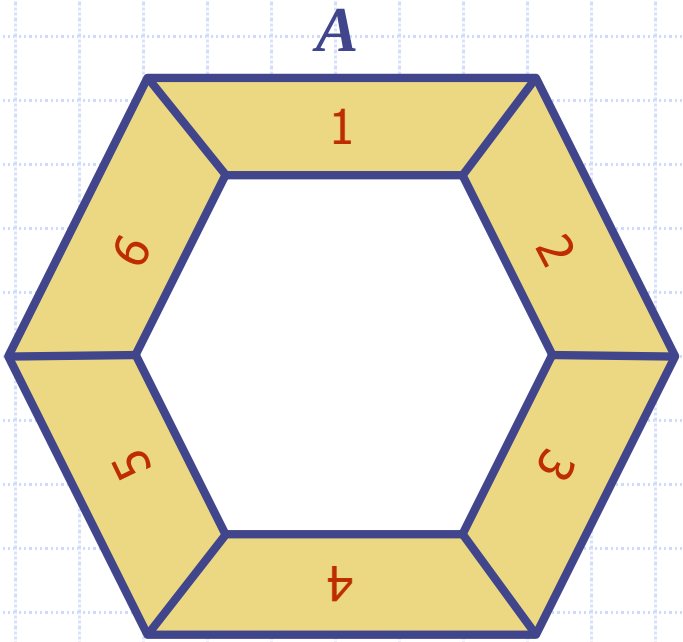
- 생일케이크에 $n > 0$개의 불켜진 양초가 **원형**으로 빙둘러 서있다
- 첫번째 양초부터 시작하여, $k > 0$개의 양초를 건너뛰어 나타나는 양초의 불을 끄고 뽑아낸다
- 그리고는 다음 양초로부터 시작하여 k개의 양초를 건너뛰어 나타나는 양초의 불을 끄고 뽑아낸다
- 원을 돌면서 양초가 **하나**만 남을 때까지 촛불 끄고 뽑아내기를 계속
- 이 마지막 양초는 내부에 특수장치가 설치되어 있어서 불이 꺼짐과 동시에 멋진 축하쇼를 펼치도록 되어 있다
- n과 k를 미리 알 경우, 원래 양초들의 원형 배치에서 특수 양초의 **위치**를 어디로 해놓아야 마지막까지 남을지 알고 싶다

응용문제: 생일 케이크 (conti.)

- **모의실행**을 통해 특수 양초의 위치를 나타내는 양의 정수를 반환하는 알고리즘을, 원형의 양초 리스트를 다음 두 가지 데이터구조로 각각 구현하고자 한다
  A. **원형배열**
  B. **원형연결리스트**
- 위 A, B 각각의 선택에 대해 아래 알고리즘을 **의사코드**로 작성하라
  - candle(n, k): buildList(n)을 호출한 후 runSimulation(L, n, k)를 수행
  - buildList(n): 요구된 데이터구조를 사용하여 크기 n의 초기 리스트 L을 구축
  - runSimulation(L, n, k): 크기 n의 리스트 L에 대해 k를 사용하여 마지막 양초만 남을 때까지 불끄기를 모의실행하고 마지막 양초의 위치를 반환
- **주의:** 모의실행을 통하지 않고 수식으로 풀어내는 방식은 불가
- **힌트:** 리스트 ADT의 기본 메쏘드 사용 가능

해결: 배열

- **데이터구조**
  - 원형배열 $A[0{:}n-1]$
    - 각 배열원소는 양초를 표현
  - 초기화
    - 각 배열원소에 $1 \sim n$을 차례로 저장: 즉, 양초의 위치



해결: 배열

- **모의실행:** 두 가지 버전 가능
  - **버전 1:** 불꺼진 양초를 **"표시"**
    - $A[0]$에서 시작하여 k개(불꺼진 양초 제외) 떨어진 양초 $A[r]$을 찾아 불을 끈다(modulo 연산 이용)
    - $A[r]$이 꺼지면 $A[r]$을 0으로 **표시**하고 남은 양초의 개수 n을 한 개 감소
    - 반복 후, n이 1이 되면 이 양초의 위치를 반환
    - 한 개의 촛불을 끄는 데 최소 k회의 modulo 연산 필요, 모두 $n-1$개의 촛불을 꺼야 하므로, **총실행시간:** $\Omega(kn)$
  - **버전 2:** 불꺼진 양초를 **"삭제"**
    - 버전 1과의 차이: $A[r]$이 꺼지면 $A[r]$을 **삭제**
    - 한 개의 양초를 제거하는 삭제 알고리즘 수행에 $O(n)$ 시간 소요, 모두 $n-1$개의 양초를 제거해야 하므로, **총실행시간:** $O(n^2)$

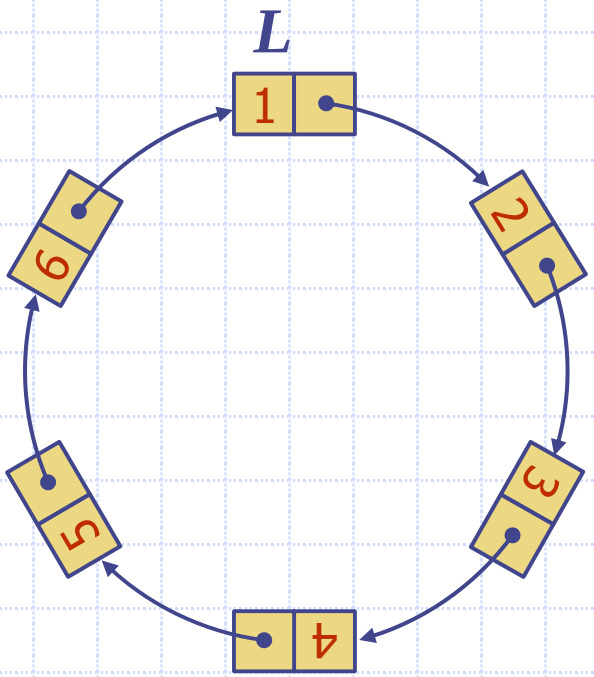
해결: 배열 (conti.)

```
Alg candle(n, k)                    {array ver.}
    input array A of size n, integer k
    output integer

1. buildList(A, n)
2. return runSimulation(A, n, k)

Alg buildList(A, n)
1. for r ← 0 to n – 1
        A[r] ← r + 1                {place index}
2. Return
```

```
Alg runSimulation(A, n, k)          {ver.1}
1. r ← 0                            {start candle}
2. while (n > 1)                    {candle remains}
        i ← 0
        while (i < k)
            r ← (r + 1) % N
            if (A[r] ≠ 0)
                i ← i + 1
        A[r] ← 0                    {remove candle}
        n ← n – 1
        while (A[r] = 0)            {reset for next round}
            r ← (r + 1) % N
3. return A[r]

Alg runSimulation(A, n, k)          {ver.2}
1. r ← 0
2. while (n > 1)                    {candle remains}
        r ← (r + k) % n
        remove(r)                   {remove candle}
3. return A[0]
```

해결: 원형연결리스트

- **데이터구조**
  - 크기 n의 원형 연결리스트 L
    - 각 노드는 양초를 표현
  - 초기화
    - 각 노드에 $1 \sim n$을 차례로 저장: 즉, 양초의 위치
- **모의실행:**
  - 첫번째 노드에서 시작하여 k개 떨어진 노드를 찾아 삭제
  - 반복 후, 남은 노드의 수가 1이 되면 그 노드에 저장된 위치를 반환
  - 한 개의 양초를 제거하는 데 k회의 연산 필요, 모두 $n-1$개의 양초를 제거해야 하므로, **총실행시간: $O(kn)$**



해결: 원형연결리스트 (conti.)

```
Alg candle(n, k)  {linked ver.}
    input integer n, k
    output integer

1. L ← buildList(n)
2. return runSimulation(L, n, k)

Alg buildList(n)
1. p ← getnode()
2. L ← p
3. p.elem ← 1      {place index}
4. for i ← 2 to n
        p.next ← getnode()
        p ← p.next
        p.elem ← i {place index}
5. p.next ← L      {make circular}
6. return L
```

```
Alg runSimulation(L, n, k)
    input circularly linked list L, integer n, k
    output integer

1. p ← L                          {start candle}
2. while (p ≠ p.next)             {candle remains}
       for i ← 1 to k – 1
           p ← p.next
       pnext ← p.next
       p.next ← (p.next).next{remove candle}
       putnode(pnext)
       p ← p.next                 {reset for next round}
3. return p.elem
```