

COSE213: Data Structure

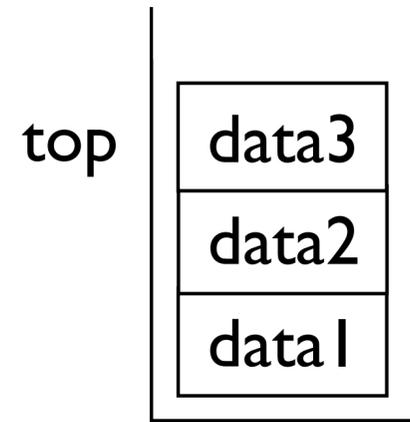
Lecture 6 - 리스트 (List)

Minseok Jeon

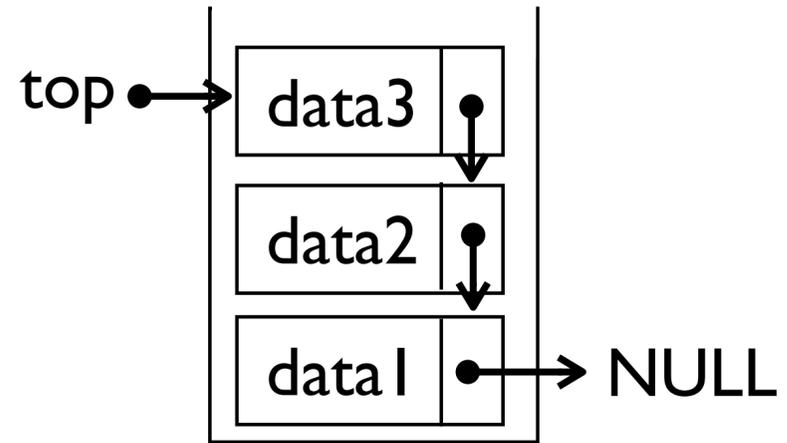
2024 Fall

Review

- 스택(Stack) 자료구조: 후입선출(LIFO) 원칙을 따르는 자료구조

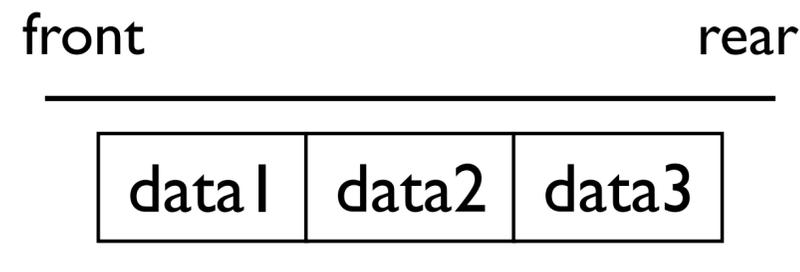


배열로 구현한 스택

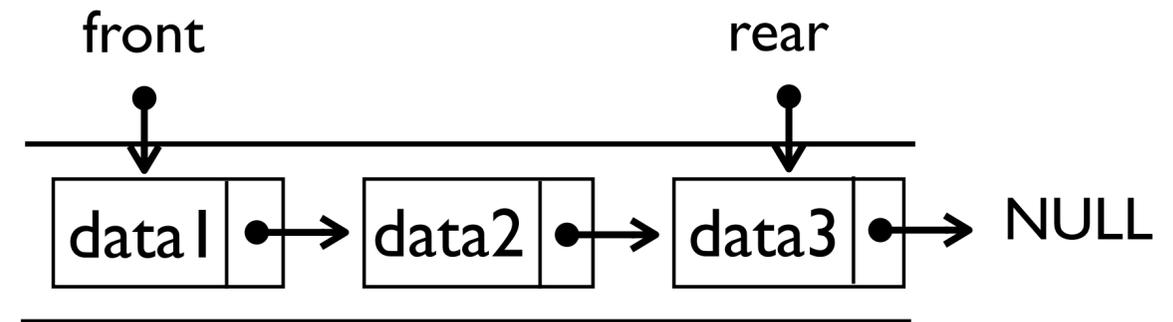


연결 리스트로 구현한 스택

- 큐(Queue) 자료구조: 선입선출 원칙을 따르는 자료구조



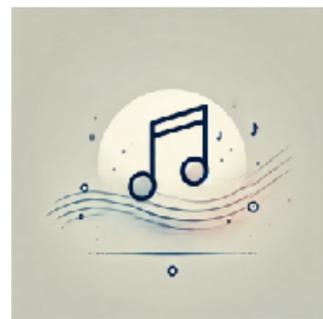
배열로 구현한 큐



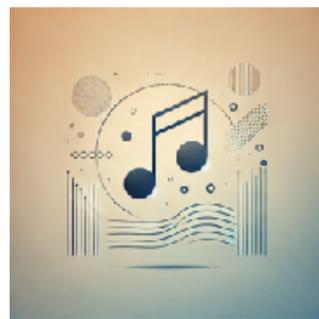
연결 리스트로 구현한 큐

문제: 플레이 리스트 관리

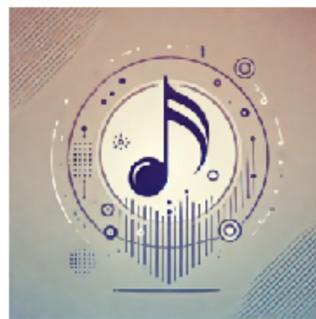
- 원하는 음악 청취를 위한 플레이 리스트 관리
 - 특징 1: 노래가 자주 추가되거나 삭제됨
 - 특징 2: 이에 따라 노래의 총 개수가 자주 바뀜
 - 특징 3: 노래의 순서가 자주 바뀜



노래 1



노래 2



노래 3



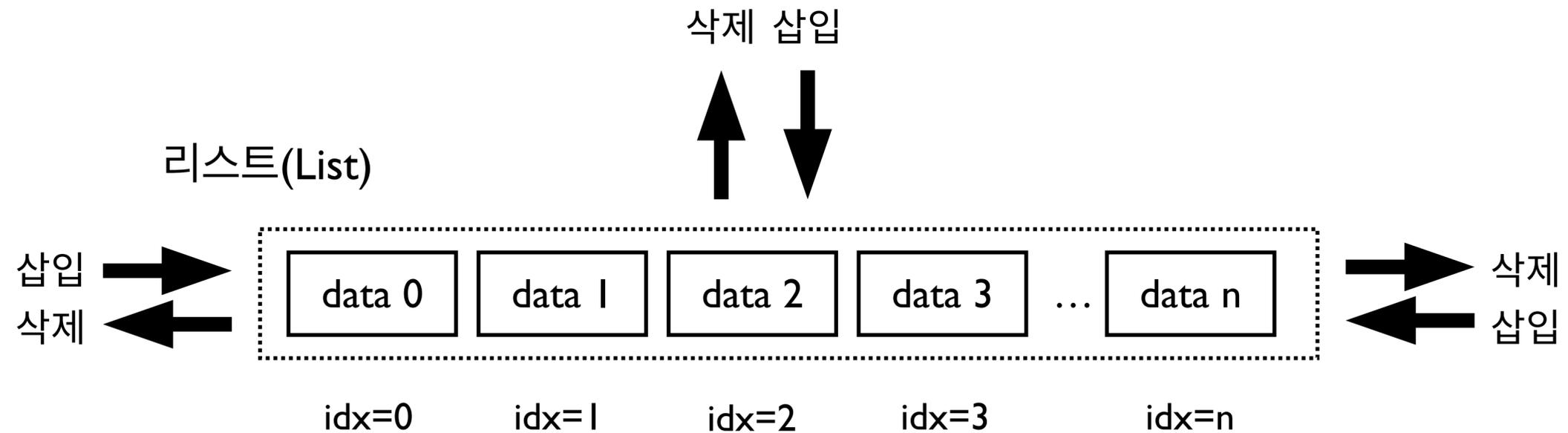
...



노래 n

해결책: 리스트

- 리스트(list) 또는 선형 리스트(linear list)는 데이터들이 순차적으로 나열되어 있는 선형 자료구조
 - 임의의 위치에 삽입, 삭제가 가능함



리스트의 구현

- 리스트의 추상 자료형:

- create : 비어있는 리스트를 생성 후 반환
- append : 리스트의 끝에 새로운 데이터를 추가함
- insert : 리스트에서 주어진 위치에 새로운 데이터를 삽입함
- deleteEntry: 리스트에서 주어진 위치에 있는 데이터를 삭제
- getEntry: 리스트에서 주어진 위치에 있는 데이터를 반환
- update: 리스트에서 주어진 위치에 있는 데이터를 새로운 데이터로 바꿈
- size : 리스트 lst 안의 데이터의 개수를 반환
- destroy : 리스트가 차지하고 있는 메모리를 해제함

```
List* create();
```

```
void append(List* list, int value);
```

```
void insert(List* list, int index, int value);
```

```
void deleteEntry(List* list, int index);
```

```
int getEntry(List* list, int index);
```

```
void update(List* list, int index, int value);
```

```
int size(List* list);
```

```
void destroy(List* list);
```

리스트의 구현

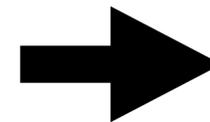
- 리스트(List)는 다음과 같은 정보를 가짐

```
typedef struct List {  
    Node* head;  
    int size;  
} List;
```

```
typedef struct Node {  
    int data;  
    struct Node* next;  
} Node;
```

- create : 비어있는 리스트를 생성 후 반환

```
procedure create()  
    list ← allocateList()  
    list.head ← NULL  
    list.size ← 0  
    return list  
end procedure
```



list

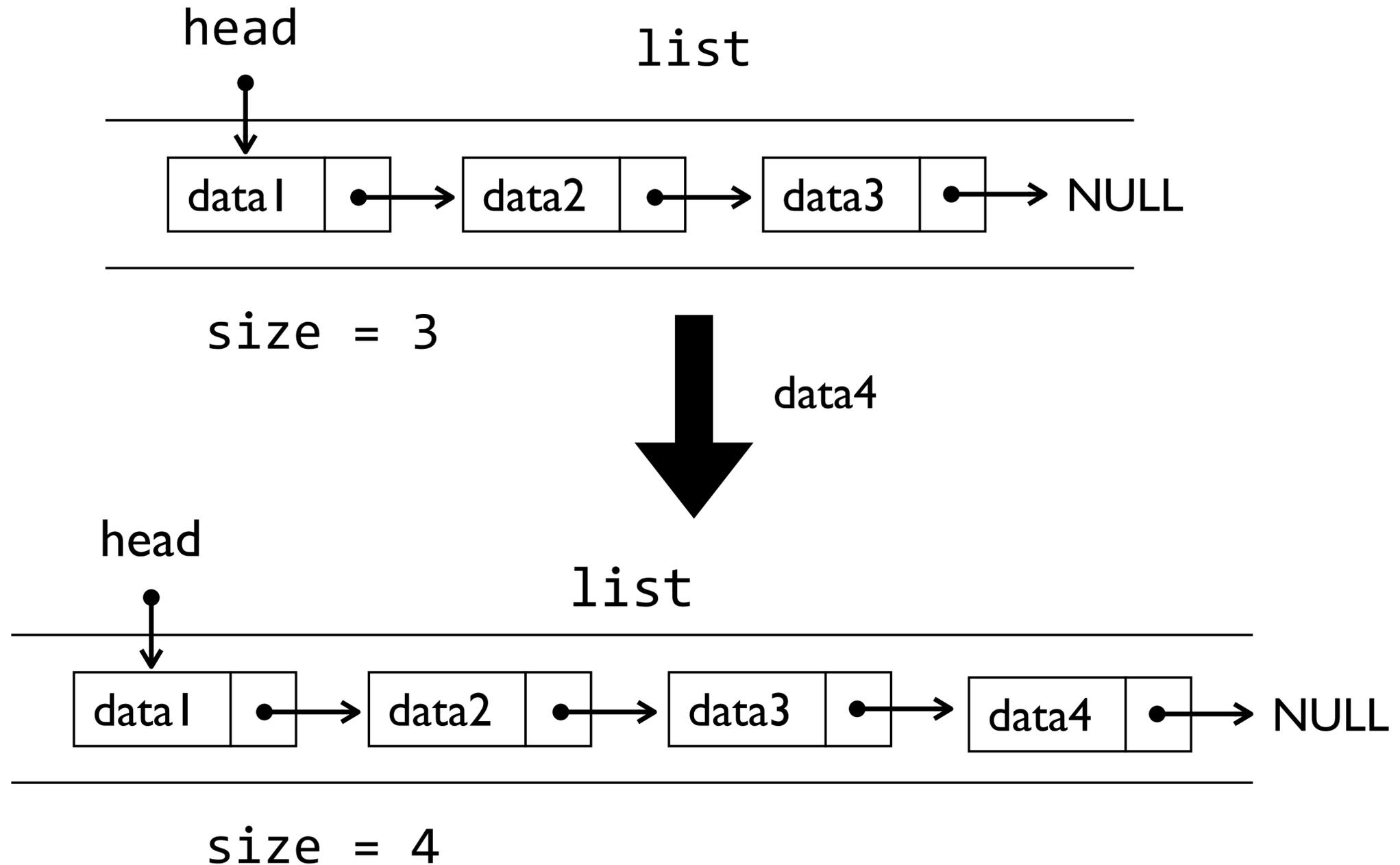


```
head = NULL  
size = 0
```

리스트의 구현

- append : 리스트의 끝에 새로운 데이터를 추가함

```
procedure append(list, data)
  node ← allocateNode()
  node.data ← data
  node.next ← NULL
  if list.head = NULL then
    list.head ← node
  end if
  else
    currentNode ← list.head
    while currentNode ≠ NULL do
      currentNode ← currentNode.next
    end while
    currentNode.next ← node
    list.size ← list.size + 1
  return list
end procedure
```



리스트의 구현

- insert : 리스트에서 주어진 위치에 새로운 데이터를 삽입함

```
procedure insert(list, index, data)
```

```
node ← allocateNode()
```

```
node.data ← data
```

```
if index = 0 then
```

```
node.next ← list.head
```

```
list.head ← node
```

```
else
```

```
currentNode ← list.head
```

```
currentIndex ← 0
```

```
while currentIndex < index-1 do
```

```
currentNode ← currentNode.next
```

```
currentIndex ← currentIndex + 1
```

```
end while
```

```
node.next ← currentNode.next
```

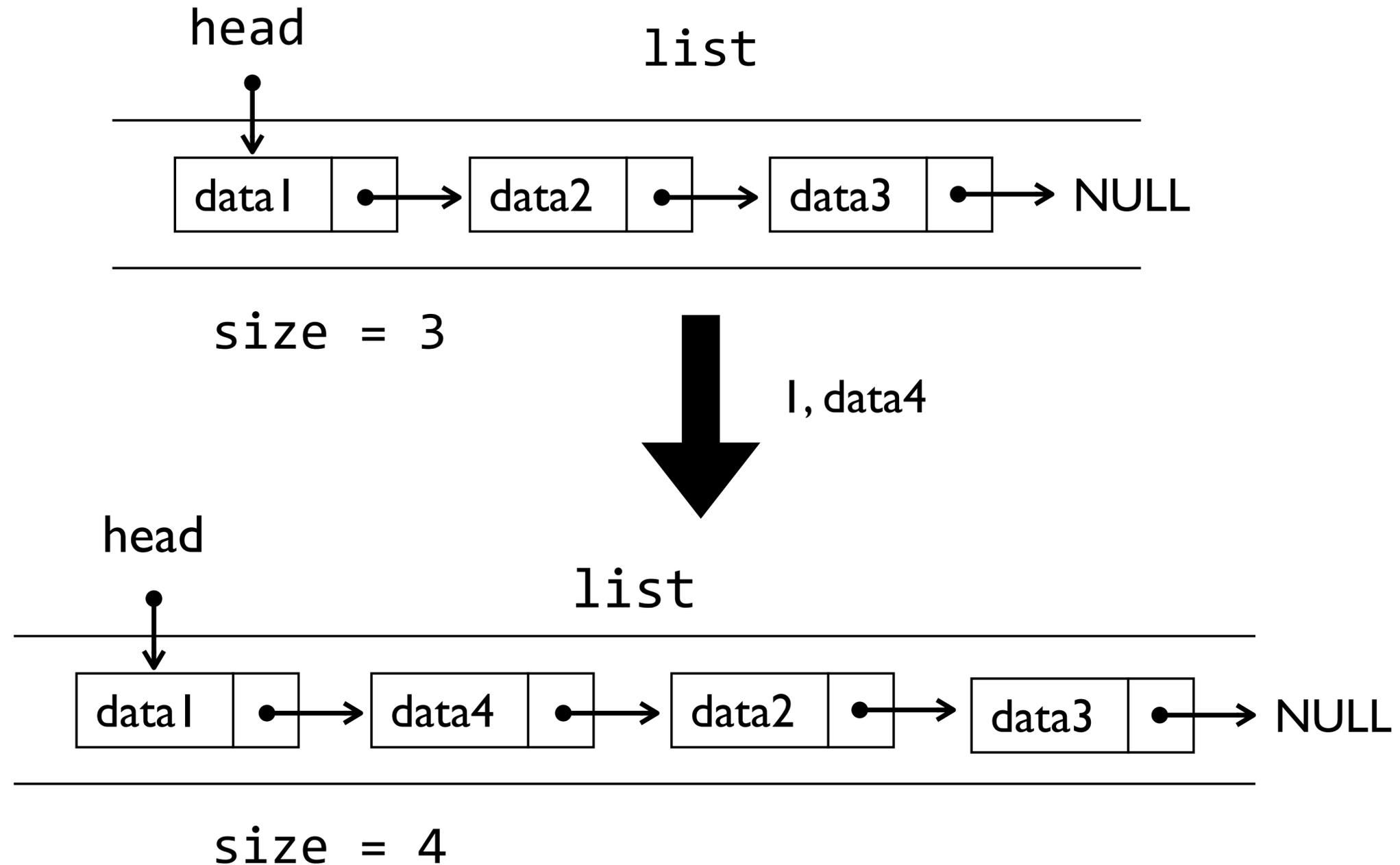
```
currentNode.next ← node
```

```
list.size ← list.size + 1
```

```
end if
```

```
return list
```

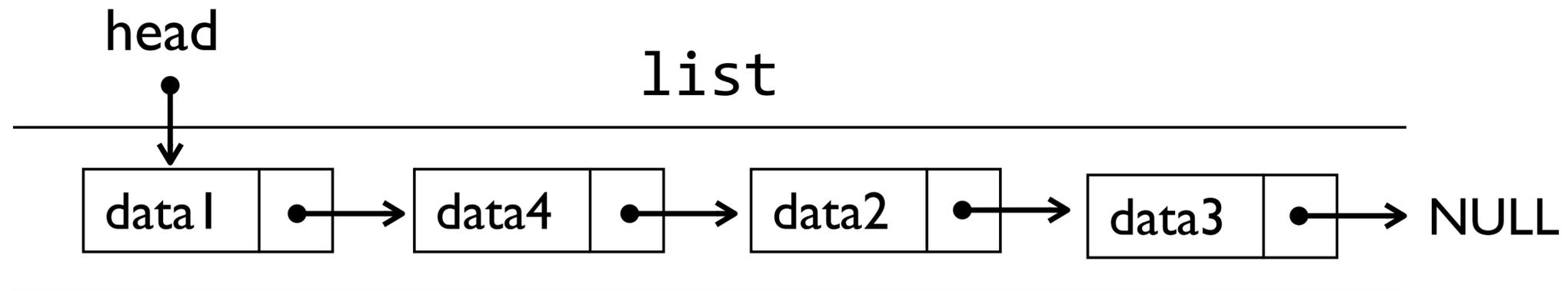
```
end procedure
```



리스트의 구현

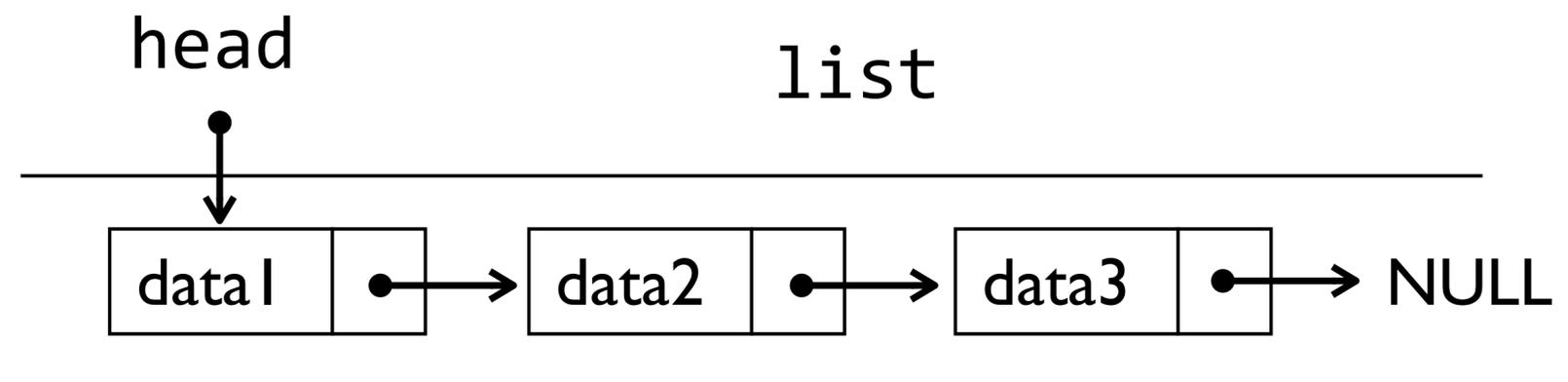
- deleteEntry: 리스트에서 주어진 위치에 있는 데이터를 삭제

```
procedure deleteEntry(list, index)
  if index < 0 or index ≥ list.size then
    return error()
  if index = 0 then
    node ← list.head
    list.head ← list.head.next
    free(node)
  else
    currentNode ← list.head
    currentIndex ← 0
    while currentIndex < index-1 do
      currentNode ← currentNode.next
      currentIndex ← currentIndex + 1
    end while
    node ← currentNode.next
    currentNode.next ← node.next
    free(node)
  end if
  list.size ← list.size - 1
  return list
end procedure
```



size = 4

index = 1

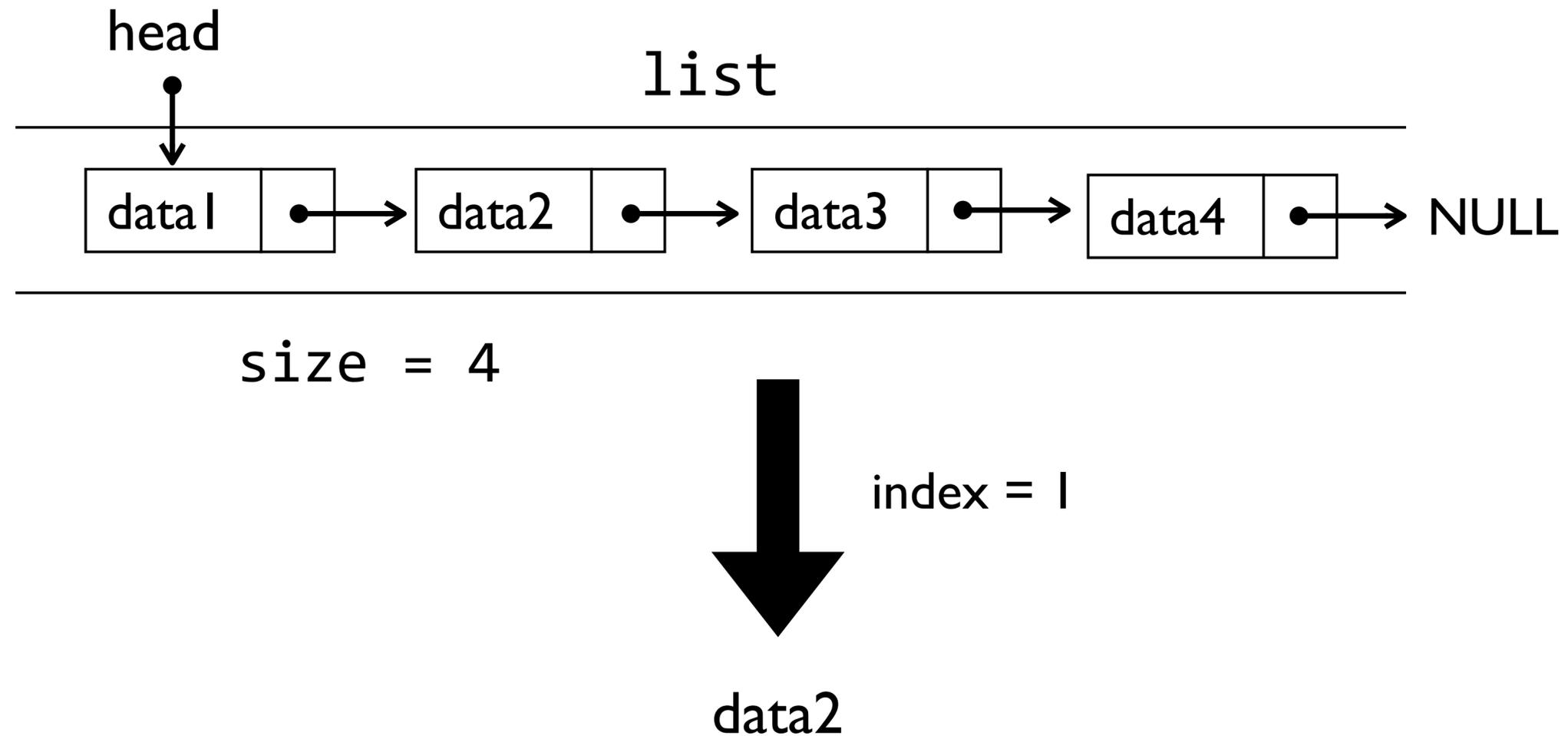


size = 3

리스트의 구현

- `getEntry`: 리스트에서 주어진 위치에 있는 데이터를 반환

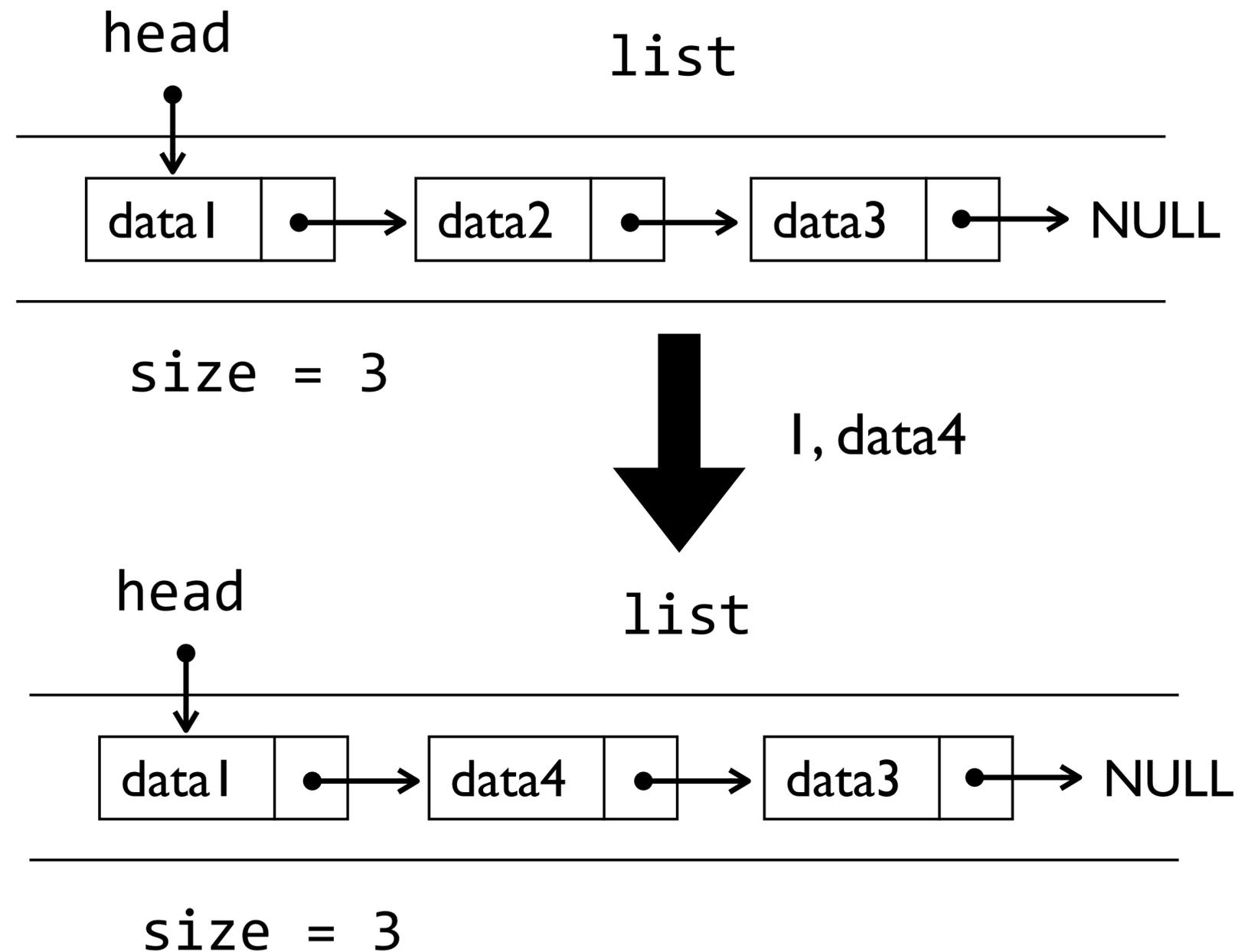
```
procedure getEntry(list, index)
  if index < 0 or index ≥ list.size then
    return error()
  currentNode ← list.head
  currentIndex ← 0
  while currentIndex < index do
    currentNode ← currentNode.next
    currentIndex ← currentIndex + 1
  end while
  return currentNode.data
end procedure
```



리스트의 구현

- update: 리스트에서 주어진 위치에 있는 데이터를 새로운 데이터로 바꿈

```
procedure update(list, index, data)
  currentNode ← list.head
  currentIndex ← 0
  while currentIndex < index do
    currentNode ← currentNode.next
    currentIndex ← currentIndex + 1
  end while
  currentNode.data ← data
  return list
end procedure
```



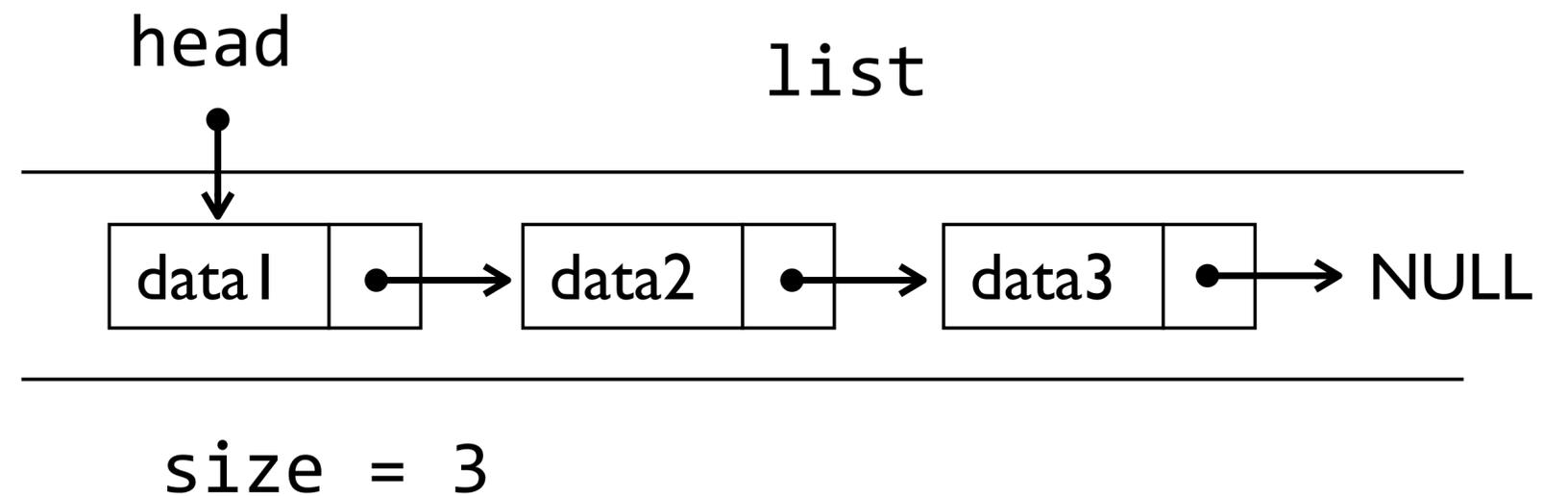
리스트의 구현

- size : 리스트 lst 안의 데이터의 개수를 반환

```
procedure size(list)
  return list.size
end procedure
```

- destroy : 리스트가 차지하고 있는 메모리를 해제함

```
procedure destroy(list)
  node ← list.head
  while node ≠ NULL do
    nextNode ← node.next
    free(node)
    node ← nextNode
  end while
  free(list)
end procedure
```



Example

```
#include <stdio.h>
#include <stdbool.h>
#include "List.h"

int main() {
    List* list = create();

    append(list, 10);
    append(list, 20);
    append(list, 30);
    printf("Entry at index 0: %d\n", getEntry(list, 0));
    printf("Entry at index 1: %d\n", getEntry(list, 1));
    printf("Entry at index 2: %d\n", getEntry(list, 2));
    insert(list, 1, 15);
    printf("list size: %d\n", size(list));
    printf("Entry at index 2: %d\n", getEntry(list, 2));

    update(list, 2, 25);
    printf("Entry at index 2: %d\n", getEntry(list, 2));
    printf("list size: %d\n", size(list));

    deleteEntry(list, 1);
    printf("Entry at index 2: %d\n", getEntry(list, 2));
    printf("list size: %d\n", size(list));

    destroy(list);

    return 0;
}
```

마무리 (Wrap-up)

- 문제:
 - 우리는 종종 자유롭게 삽입 삭제를 할 수 있는 선형 자료구조가 필요함 (예: 플레이 리스트)
- 해결책:
 - 리스트(list): 데이터들이 순차적으로 나열되어 있는 선형 자료구조이고 임의의 위치에 데이터 삽입, 삭제가 가능함
 - 리스트의 추상 자료형 (ADT):
 - create : 비어있는 리스트를 생성 후 반환
 - append : 리스트의 끝에 새로운 데이터를 추가함
 - insert : 리스트에서 주어진 위치에 새로운 데이터를 삽입함
 - deleteEntry: 리스트에서 주어진 위치에 있는 데이터를 삭제
 - ...