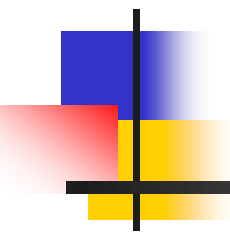


C Programming (CSE2035)

(Chap12. Lists 1)

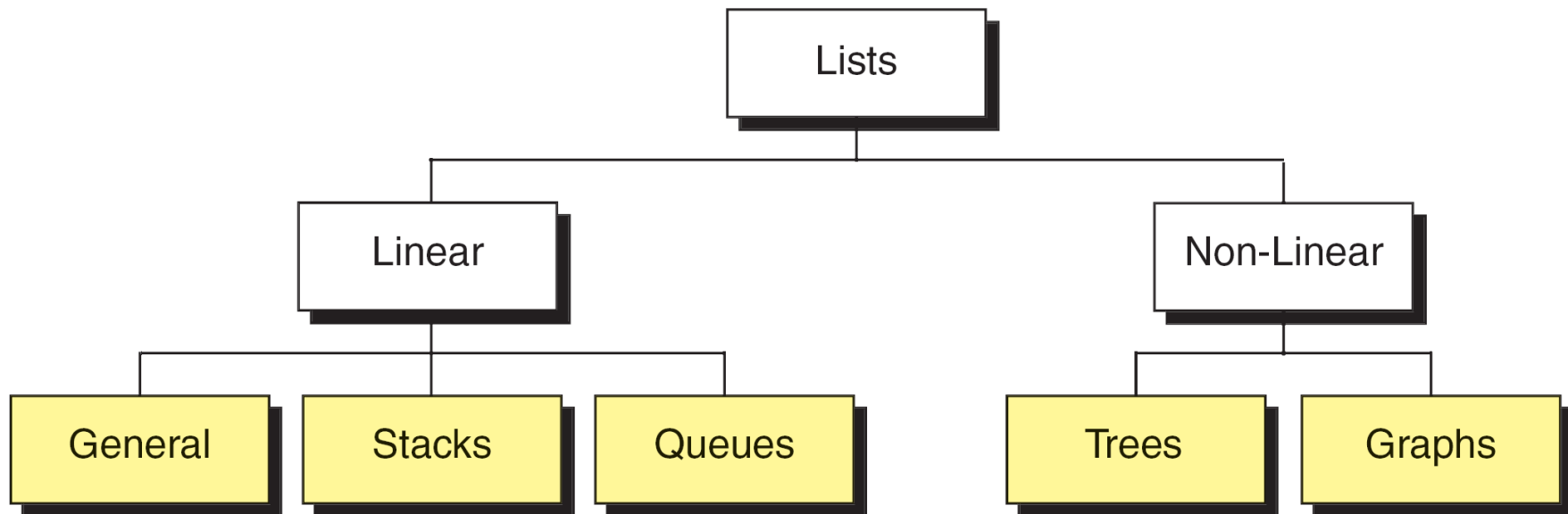


Sungwon Jung, Ph.D.

Bigdata Processing & DB LAB
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Lists

- List는 연관된 data들의 collection이다.



List Implementations

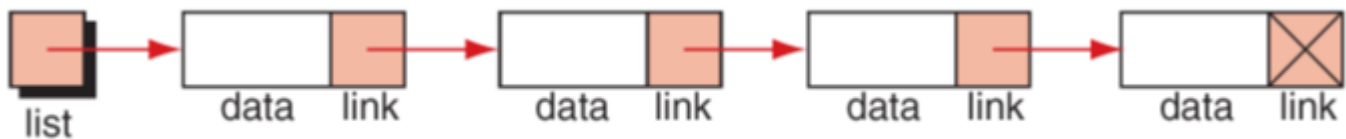
■ Array를 이용한 구현

- 장점: 원소의 Searching이 쉽다.
- 단점: 원소의 삽입 및 삭제가 쉽지 않다.

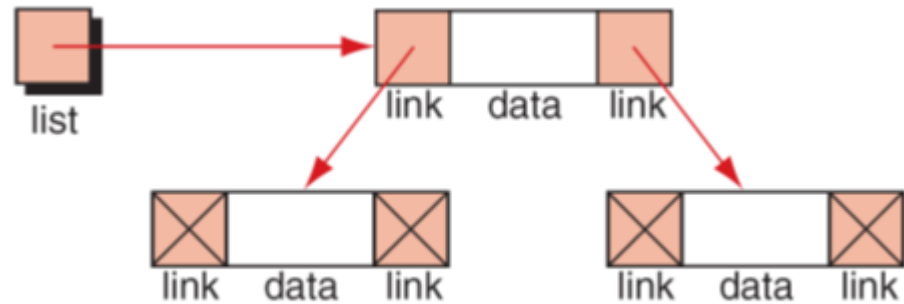
Array 구조:
arr[]

| | |
|--------|----|
| arr[3] | 1 |
| arr[2] | 5 |
| arr[1] | 19 |
| arr[0] | 52 |

■ Linked list를 이용한 구현



(a) Linear list



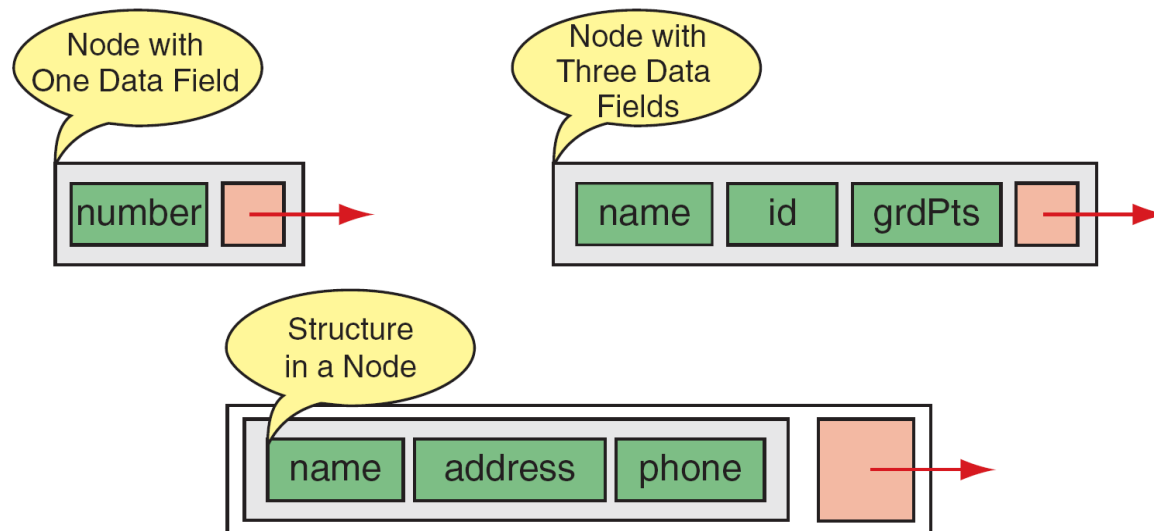
(b) Non-linear list



(c) Empty list

Linked List Node Structures

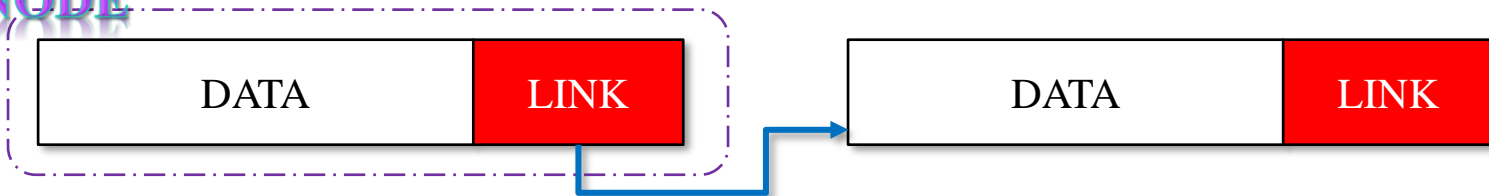
- Linked List는 항상 head pointer를 가진다.
- 어느 용도로 list를 사용하느냐에 따라 pointer를 더 사용할 수도 있다.



Linked List Structure

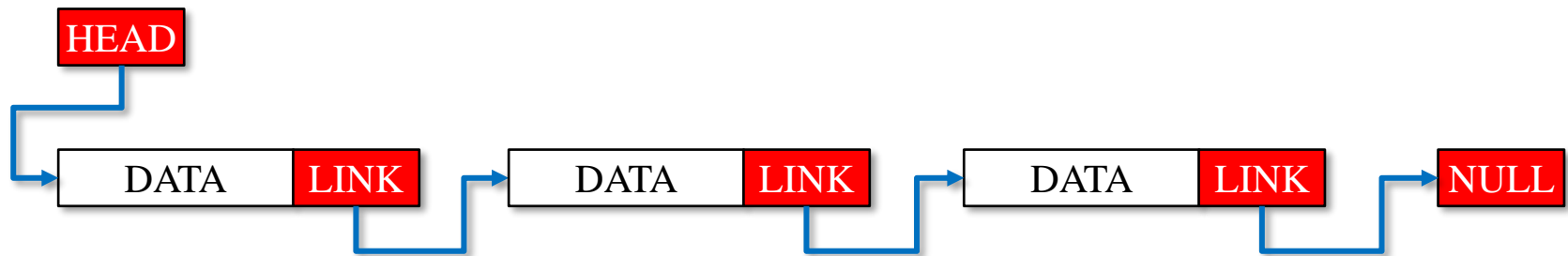
- **Linked list** 는 다음 정보(**node**)를 가리키는 포인터를 가지고 있는 **data** 구성방식의 한 형태이다. 포인터로 마치 체인과 같이 연결되어 있는 **list** 라고 해서 **linked list**라 한다.
- **Linked list**의 **node**는 **data** 부분과 **link** 부분의 두 **parts**로 구성된다. 또 **node**의 추가와 삭제가 자유로워 메모리를 효율적으로 활용할 수 있다. 고로 데이터의 추가 삭제가 잦고 크기가 정해지지 않은 경우 유용하다.

NODE



Linked List Structures

- List의 첫 번째 node를 가리키는 포인터를 헤드 포인터라고 한다. linked list의 node는 다음 node를 가리킨다.
- 마지막 node의 포인터는 NULL 값을 가진다.



< Linked list 시작과 끝 >

Linked List Structures

- **Linked list**를 C언어로 표현하면 다음과 같다.

```
typedef struct _NODE{
    DATATYPE      DATA;
    struct _NODE *LINK;
}NODE;
```



- 만약 **data**가 **integer** 하나인 **linked list**를 만든다면 **node**와 헤드 포인터는 다음과 같이 표현된다.

```
typedef struct _NODE {
    int      Data;
    struct _NODE *LINK;
}NODE;

NODE *head;
```

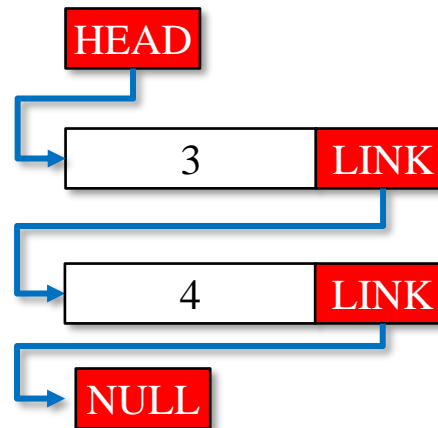
Linked List Structures

```
typedef struct _NODE {
    int          Data;
    struct _NODE *LINK;
}NODE;

NODE *head;
```

- 위 node를 가지고 정수 3, 4 순서대로 저장하는 Linked list를 동적으로 생성해보자.

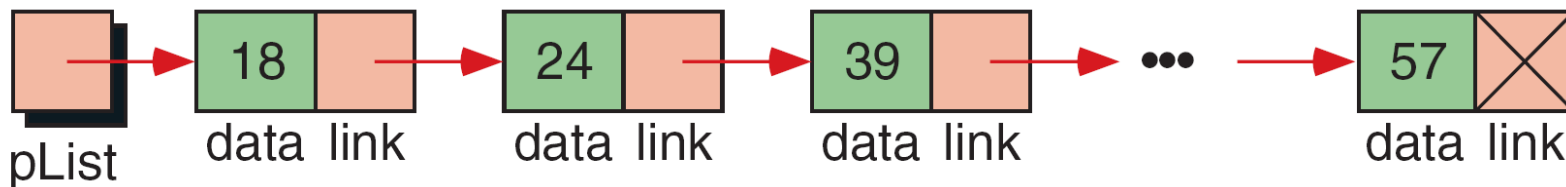
```
head = malloc(sizeof(NODE));
head->data = 3;
head->link = malloc(sizeof(NODE));
head->link->data = 4;
head->link->link = NULL;
```



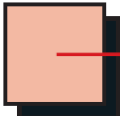
Insert a Node

■ Linear 리스트에 node 삽입하기

- 새로운 node를 위해 메모리를 할당한다.
- 삽입할 위치를 선택: 정확한 위치를 알기 위해 삽입할 위치의 이전 node의 위치를 파악
- 새로 삽입된 node가 삽입된 위치 뒤의 node를 가리키도록 한다.
- 새로 삽입된 node 이전의 node가 새로운 node를 가리키도록 한다.



pPre  Add to empty list or add at beginning of list

pPre  Add in middle of list or add at end of list

Insert a Node

- `NODE* insertNode(NODE *pList, Node *pPre, DATA item)`
 - 원래의 list인 `pList`를 가리키는 포인터와 삽입하고자 하는 node의 바로 이전 node인 `pPre`를 가리키는 포인터, 그리고 새로운 node의 data인 `item`을 입력 받는다.
 - 새로운 노드를 `pList`에 삽입한다.

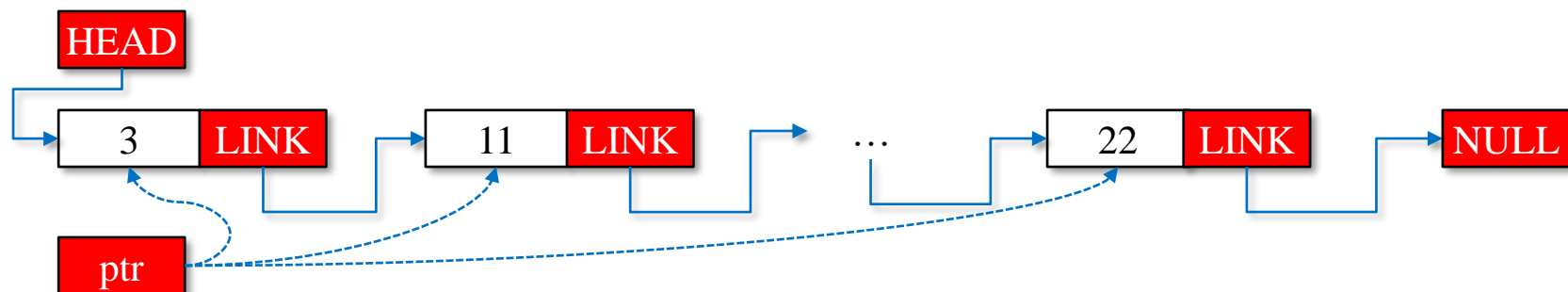
```
NODE* insertNode(NODE *pList, NODE *pPre, DATA item) {  
    NODE *pNew;  
    pNew = (NODE*)malloc(sizeof(NODE));  
    pNew->data = item;  
    if (pPre == NULL) {  
        pNew->link = pList;  
        pList = pNew;  
    } else {  
        pNew->link = pPre->link;  
        pPre->link = pNew;  
    }  
    return pList;  
}
```

새로운 node에 공간을 할당

삽입할 node의 이전 node가 NULL인 경우는 가장 처음에 새로운 node를 삽입할 경우이다.

Traversing linked lists

- Linked list를 탐색하는 방법
 - 배열의 경우 index가 존재해서 이를 이용하여 데이터를 탐색한다.
 - Linked list의 경우 헤드에서 부터 순차적으로 link를 따라가 탐색한다.



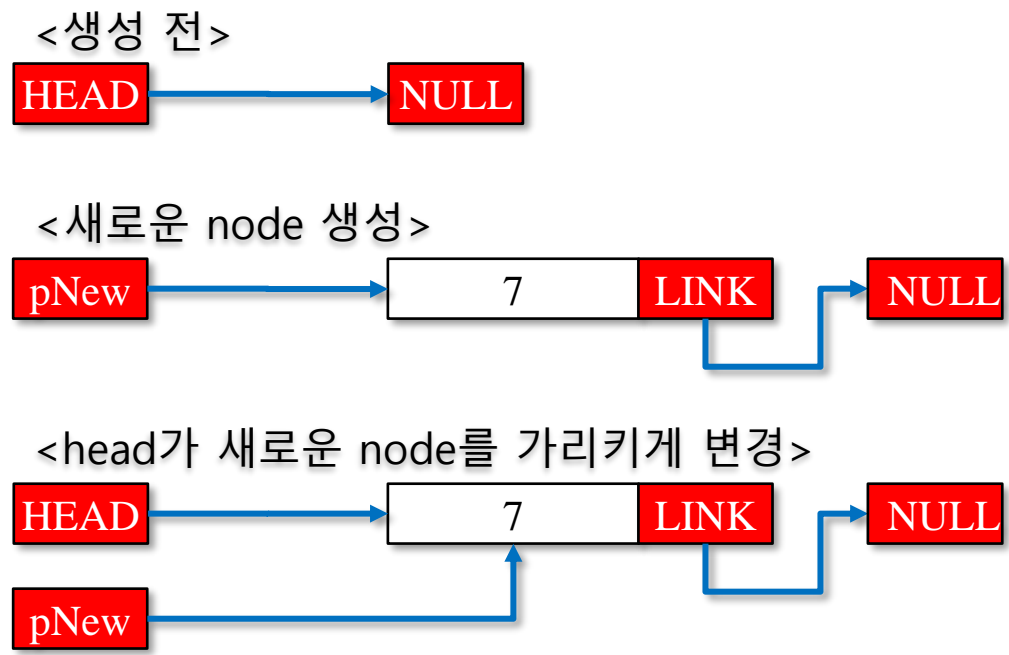
- 위 예제에서 data가 22인 node를 찾고 싶다면 아래와 같이 동작한다.

```
NODE *ptr;  
  
ptr = head;  
while(ptr != NULL) {  
    if(ptr->data == 22) break;  
    ptr = ptr->LINK;  
}
```

Insert into Empty List

■ 빈 list에 node 삽입하기

- 비어있는 list에서 head는 NULL을 가리키고 있다.
- 새로운 node를 생성한 후, head가 이 node를 가리키게 만든다.



```

NODE *pNew;

pNew = malloc(sizeof(NODE));
pNew->LINK = HEAD;
/* pList == HEAD */

pNew->DATA = 7;

HEAD = pNew;
  
```

Insert at Beginning

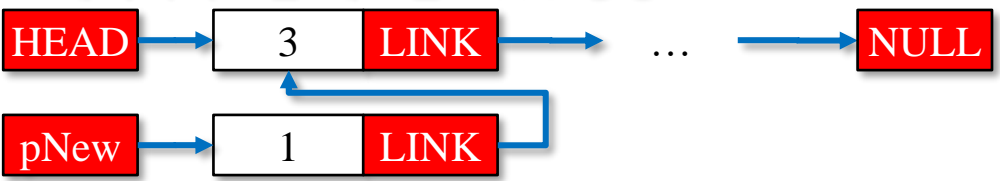
List의 처음에 node 삽입하기

- 이미 존재하는 list는 유지한 채 처음, 즉 head가 가리키는 node를 새로 생성하는 node로 교체하는 방법이다.
- 새로운 node는 이전의 첫 node를 가리키도록 한다.

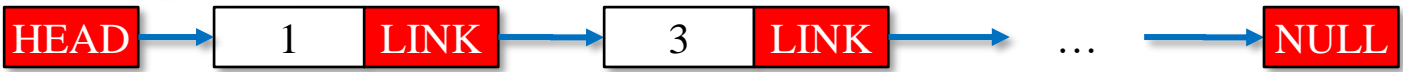
<원래 list>



<데이터가 1인 새로운 node 생성>



<head를 새로운 node로 이동>



```

NODE *pNew;

pNew = malloc(sizeof(NODE));
pNew->LINK = HEAD;
/* pList == HEAD */

pNew->DATA = 1;

HEAD = pNew;
    
```

Insert at Middle

- list의 중간에 node 삽입하기
 - 특정 위치에 node를 삽입하기 위해서는 현재 위치를 가리키는 node (즉 바로 전 node)를 알고 있어야 한다. 새로운 node를 생성한 후 생성된 node의 링크를 전 node의 링크로 하고, 전 node의 링크를 새로운 node로 바꾼다.

