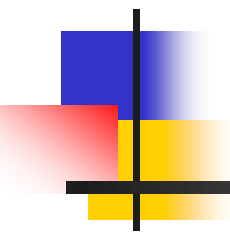


# **C Programming (CSE2035)**

## **(Chap12. Lists 2)**



---

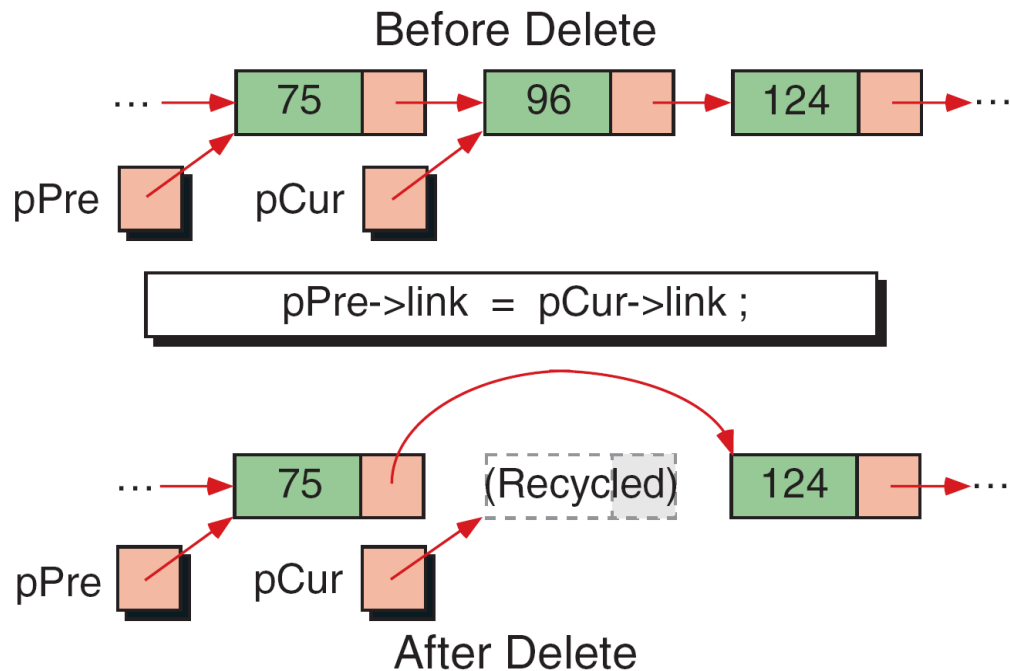
**Sungwon Jung, Ph.D.**

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

# Delete a Node

## ■ Linear List로부터 node 삭제하기

- 삭제할 위치를 선택: 정확한 위치를 알기 위해 삭제할 node와 이전 node의 위치를 파악
- 삭제할 node의 이전 node가 삭제할 node의 다음 node를 가리키도록 한다.
- 삭제된 node는 free시킨다.



## Delete a Node

- `NODE* deleteNode(NODE *pList, Node *pPre, Node* pCur)`
  - `pList`로부터 하나의 `node`를 삭제하는 함수로 해당 `node`가 삭제된 `list`를 `return`
  - Parameter: `pList`는 `list`의 `head`를 가리키는 포인터  
`pPre`는 삭제하려는 `node`의 이전 `node`를 가리키는 포인터  
`pCur`은 삭제하려는 `node`를 가리키는 포인터

```
NODE* deleteNode(NODE* pList, NODE* pPre, NODE* pCur) {  
    if (pPre == NULL) {  
        pList = pCur->link;  
    } else {  
        pPre->link = pCur->link;  
    }  
    free(pCur);  
    return pList;  
}
```

맨 처음 `node`를 삭제할 경우

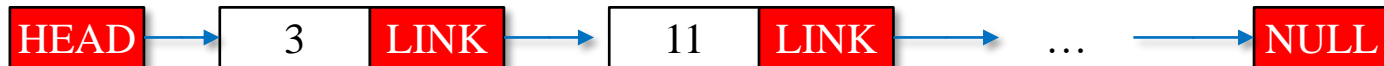
맨 처음 `node`가 아닌 `node`를 삭제할 경우

# Delete a Node

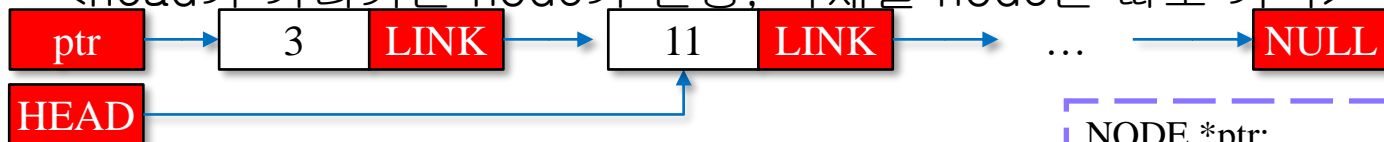
## list의 처음 node 삭제하기

- 첫 node가 리스트로부터 삭제되면 헤더가 가리키는 node가 변경되어야 한다. 헤더가 가리키는 node는 첫 node가 가리킨 node, 즉 리스트에서 두 번째 node가 된다. (만약 존재하지 않는다면 당연히 NULL이 된다.)

<원래list>



<head가 가리키는 node가 변경, 삭제될 node는 따로 기억>



<node 삭제>



```
NODE *ptr;
```

```
ptr = head;
```

```
head = head->link;
```

```
free(ptr);
```

# Delete a Node

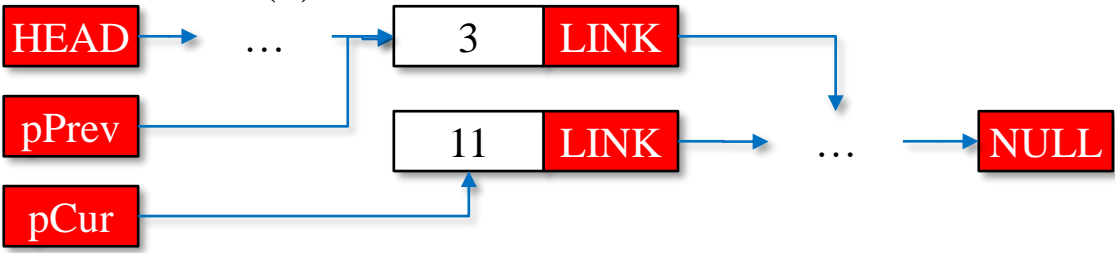
## ■ 일반적인 list에서 node 삭제하기

- 삭제 될 node를 가리키는 node, 즉 삭제될 node의 이전 node를 기억한다.
- 이전 node의 LINK를 삭제될 node의 LINK로 대체한다.

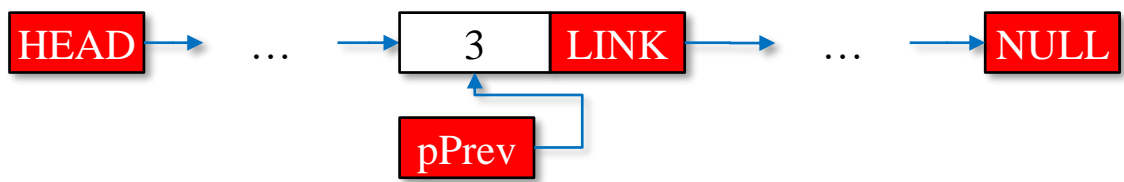
<원래list>



<이전 node(3)의 LINK를 변경>



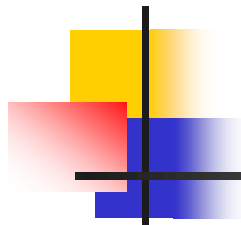
<node삭제>



```

NODE *pPrev, *pCur;
pCur = pPrev = head;

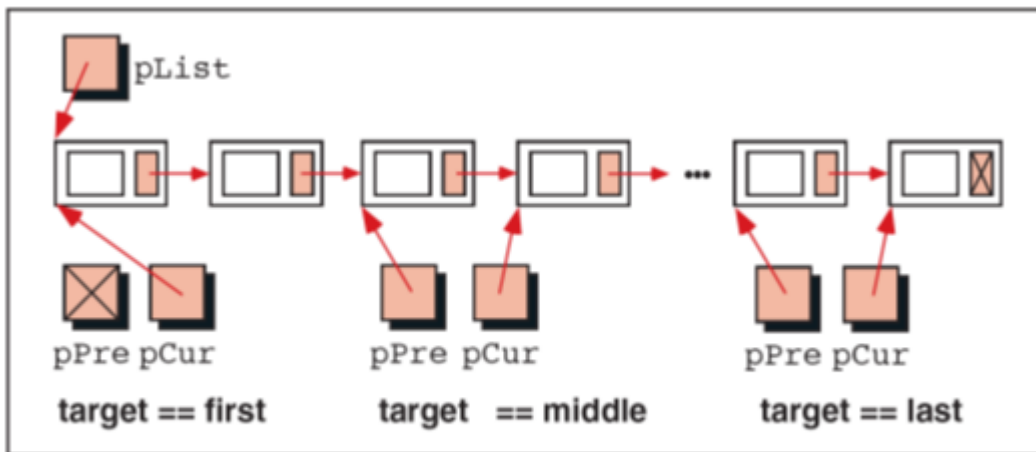
if(pCur == NULL){
    printf("The list is empty\n");
}else{
    while(pCur->link != NULL &&
        pCur->data != 11) {
        pPrev = pCur;
        pCur = pCur->link;
    }
    pPrev->link = pCur->link;
    free(pCur);
}
  
```



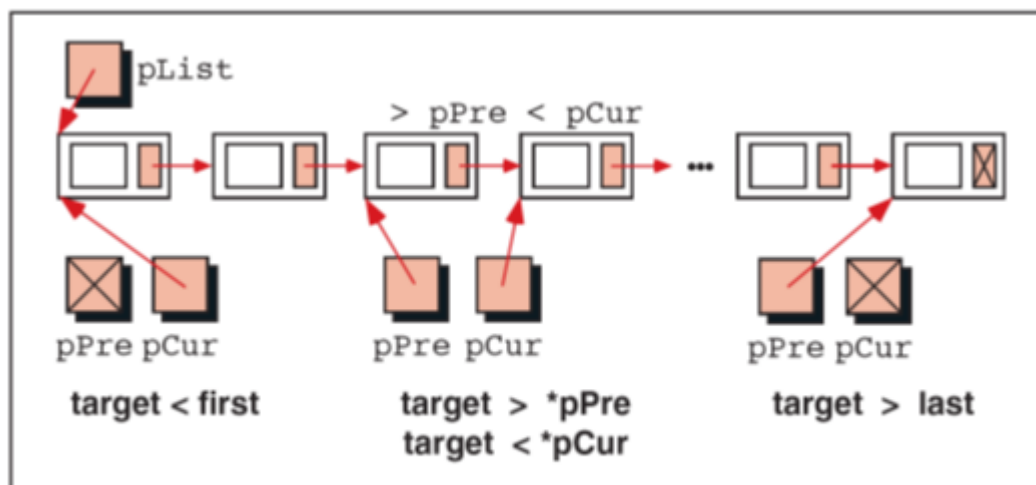
# Search a Node in a Linear List

Condition	pPre	pCur	Return
target < first node	NULL	first node	0
target == first node	NULL	first node	1
first < target < last	largest node < target	first node > target	0
target == middle node	node's predecessor	equal node	1
target == last node	last's predecessor	last node	1
target > last node	last node	NULL	0

# Search a Node in a Linear List



Successful Searches (Return *true*)



Unsuccessful Searches (Return *false*)

# Search a Node in a Linear List

```
bool searchList (NODE*  pList, NODE**  pPre,
                NODE**  pCur, KEY_TYPE target)
{
    // Local Declarations
    bool found = false;
    // Statements
    *pPre = NULL;
    *pCur = pList;

    // start the search from beginning
    while (*pCur != NULL && target > (*pCur)->data.key)
    {
        *pPre = *pCur;
        *pCur = (*pCur)->link;
    } // while

    if (*pCur && target == (*pCur)->data.key)
        found = true;
    return found;
} // searchList
```

```
typedef int KEY_TYPE;

typedef struct {
    KEY_TYPE key;
    ...
}DATA;

typedef struct nodeTag {
    DATA data;
    struct nodeTag* link;
}NODE;
```