# CHAPTER 4

# Linked Lists

# LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists

- Sequential representation

  연속적인 데이터들이 일정 거리로 떨어져있다

  - Successive items are located a fixed distance apart
  - Adequate for the tasks:
    - Accessing an arbitrary node in a **table**
    - Insertion or deletion of **stack** and **queue** elements

    ) 배열을 사용

- Disadvantage
  - For **ordered lists**, operations such as insertion and deletion of arbitrary elements become expensive
  - E.g. To add the word GAT the following list :

(BAT, CAT, EAT, FAT, HAT, JAT, LAT, MAT, OAT, PAT, RAT, SAT, VAT, WAT)

⇒ GAT를 넣으려고 하는 원소들을 shift를 해주어야함

- ***Linked*** representations
  - Items may be placed anywhere in memory
  - A linked list is comprised of nodes
  - Nodes
    - data fields + link(or pointer) fields
  - Link
    - Store the address or location of the next element in that list

For any i,
**data[i]** and **link[i]** together comprise a node

**first** = 8

data[8] = BAT

link[8] = 3

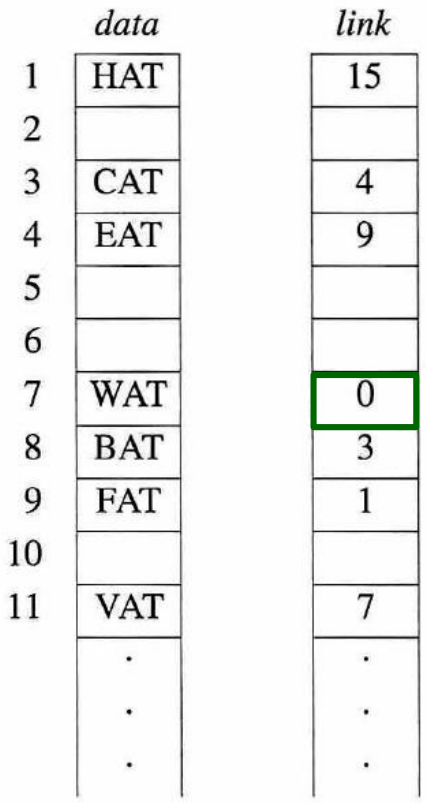
data[3] = CAT

link[3] = 4

data[4] = EAT

data[link[4]] = FAT

…




end

↳ 데이터의 끝을 나타냄

**Figure 4.1:** Nonsequential list-representation using two arrays

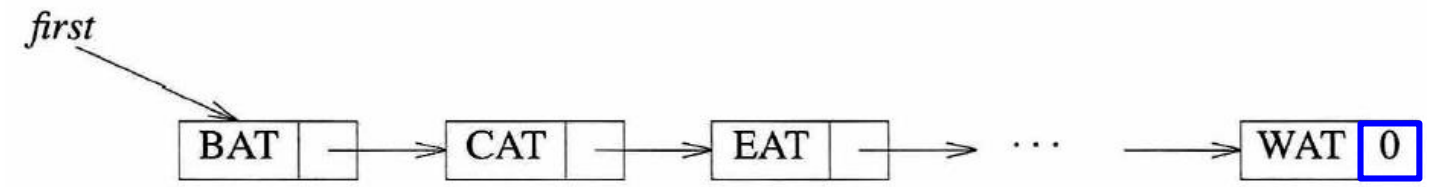


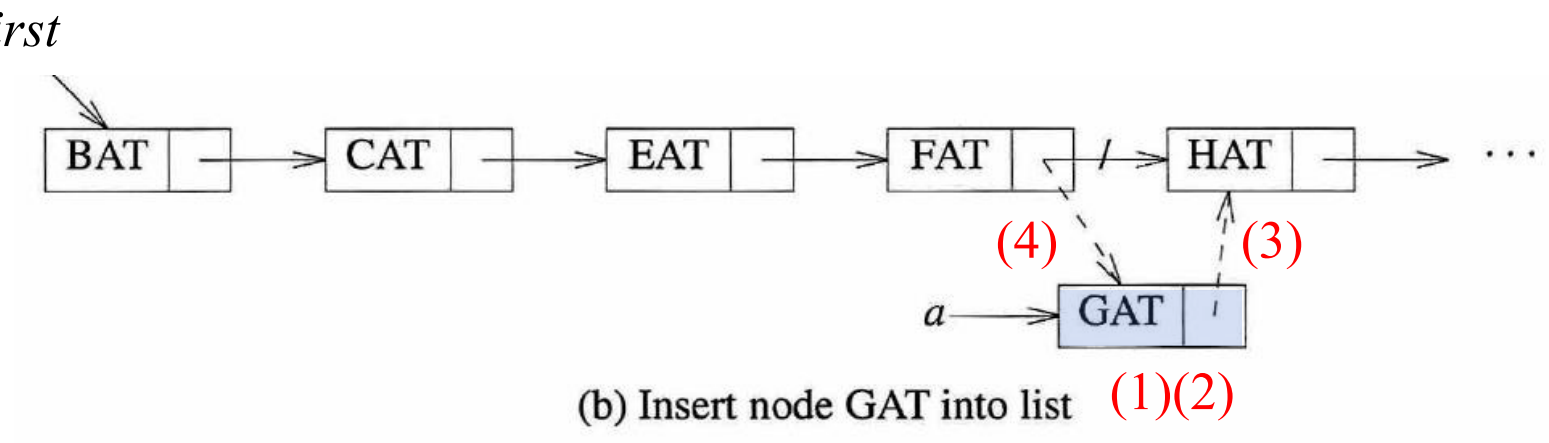

**Figure 4.2:** Usual way to draw a linked list

first의 link만 알면됨!

The linked structures are called singly linked lists or chains.

Insertions and deletions : …

- Linked list: Insert (e.g. GAT):
  (1) Get a node *a* that is currently unused
  (2) Set the *data* field of *a* to GAT
  (3) Set the *link* field of *a* to point to the node (HAT)
  (4) Set the *link* field of the node (FAT) to *a*




(b) Insert node GAT into list

(3), (4) 바뀌면 안됨!
HAT을 가리키는 사슬을 잃어버리게됨

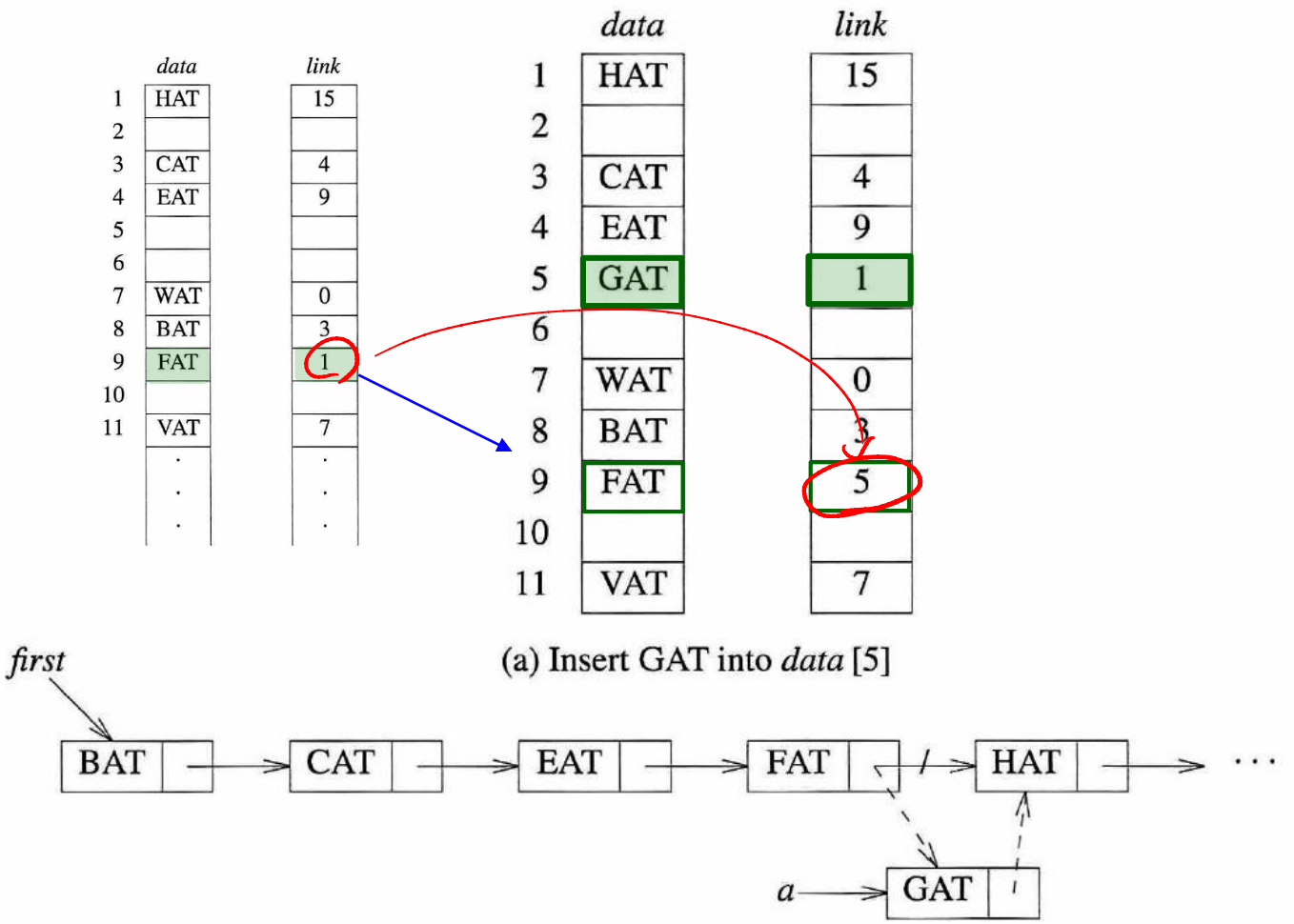



**Figure 4.3:** Inserting into a linked list

- Linked list: **Delete** (e.g. GAT)
  - ① – Find the element (FAT) that immediately precedes GAT
  - ② – Set its link field to point to the position of HAT

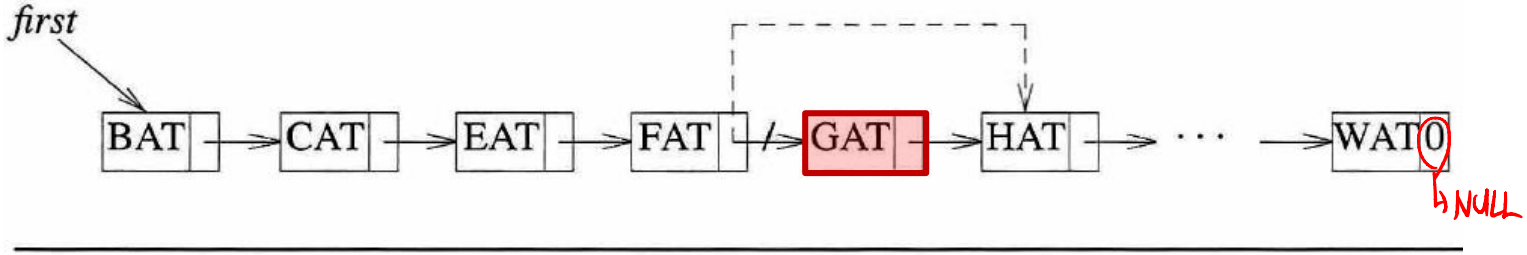



**Figure 4.4:** Delete GAT

# LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists

- Ex 4.1 [List of words]:
  - Define a node structure for the list

```
typedef struct listNode *listPointer;
typedef struct listNode {
        char data [4];
        listPointer link;
};
```

구조체 포인터

typedef struct listNode {
    char data[4];
    struct listNode *link;
} listNode;

  - Create a new empty list

```
listPointer first = NULL;
```

  - Test for an empty list

empty 이면 참이다.

```
#define IS_EMPTY( first ) ( !(first) )
```

  - Allocation; to create new nodes

```
MALLOC( first, sizeof(*first) );
```

  - To place the word BAT into the list

```
strcpy( first -> data, "BAT");
first -> link = NULL;
```

```
…
if( IS_EMPTY( first ) )
        …
```

```
#define MALLOC(p,s) \
if (! ( (p) = malloc (s) ) ) { \
  fprintf(stderr, "Insufficient memory"); \
  exit(EXIT_FAILURE);\
}
```

OVR

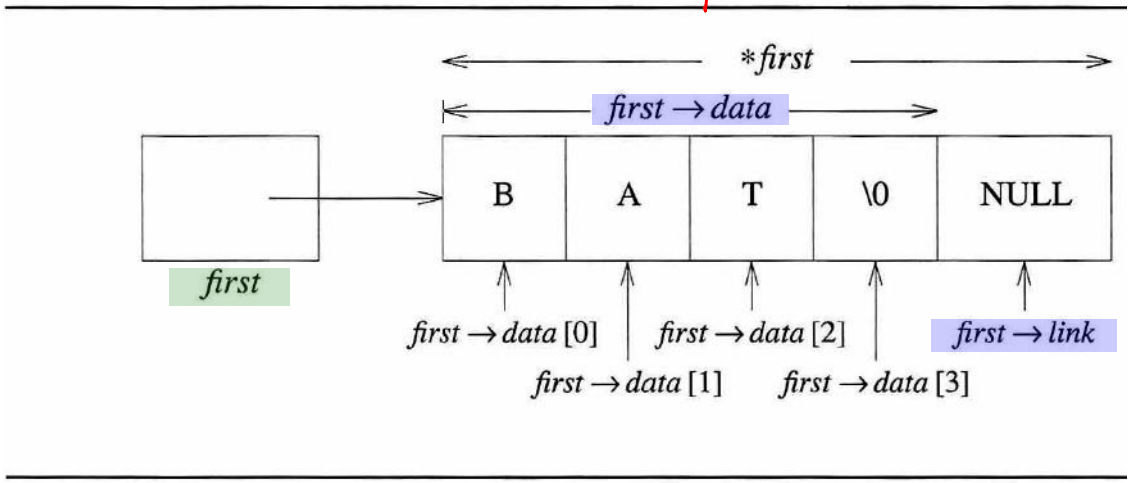



**Figure 4.5:** Referencing the fields of a node

```
typedef struct listNode *listPointer;
typedef struct listNode {
        char data [4];
        listPointer link;
};
```

- Ex 4.2 [**Two-node linked list**]
  - To create a linked list of integers : …

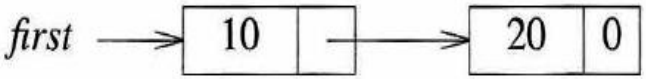



**Figure 4.6:** A two-node list

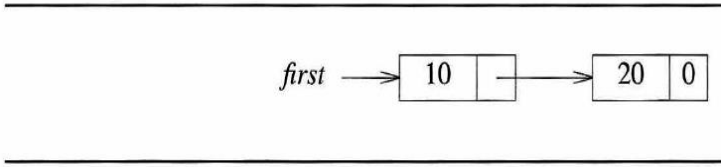



**Figure 4.6:** A two-node list

```
listPointer create2()
{
     listPointer  first, second;

     MALLOC(first, sizeof(*first));
     MALLOC(second, sizeof(*second));

     second->link = NULL;
     second->data = 20;

     first->data = 10;
     first-> link = second;

     return first;
}
```

- Ex 4.3 [**List Insertion**]: insert (&first, x );
  - Let *first* be a pointer to a linked list
  - Assume that we want to insert a node with a data field of 50 after some arbitrary node *x*

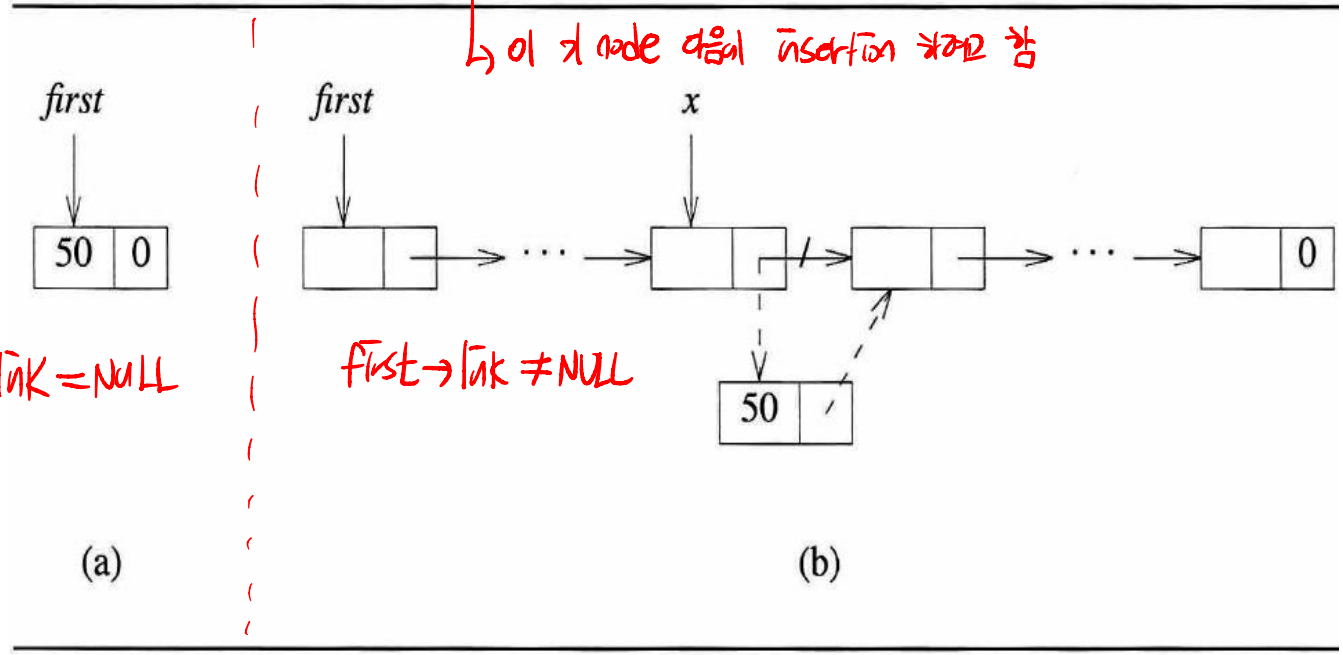



**Figure 4.7:** Inserting into an empty and nonempty list

```
void insert(listPointer *first, listPointer x)
{
    listPointer temp;
    MALLOC(temp, sizeof(*temp));
    temp -> data = 50;
    if (*first) {     // noempty list
        temp->link = x->link;
        x->link = temp;
    }
    else {            // empty list
        temp->link = NULL;
        *first = temp;
    }
}
```

**Program 4.2:** Simple insert into front of list

Function call:
insert (&first, x );

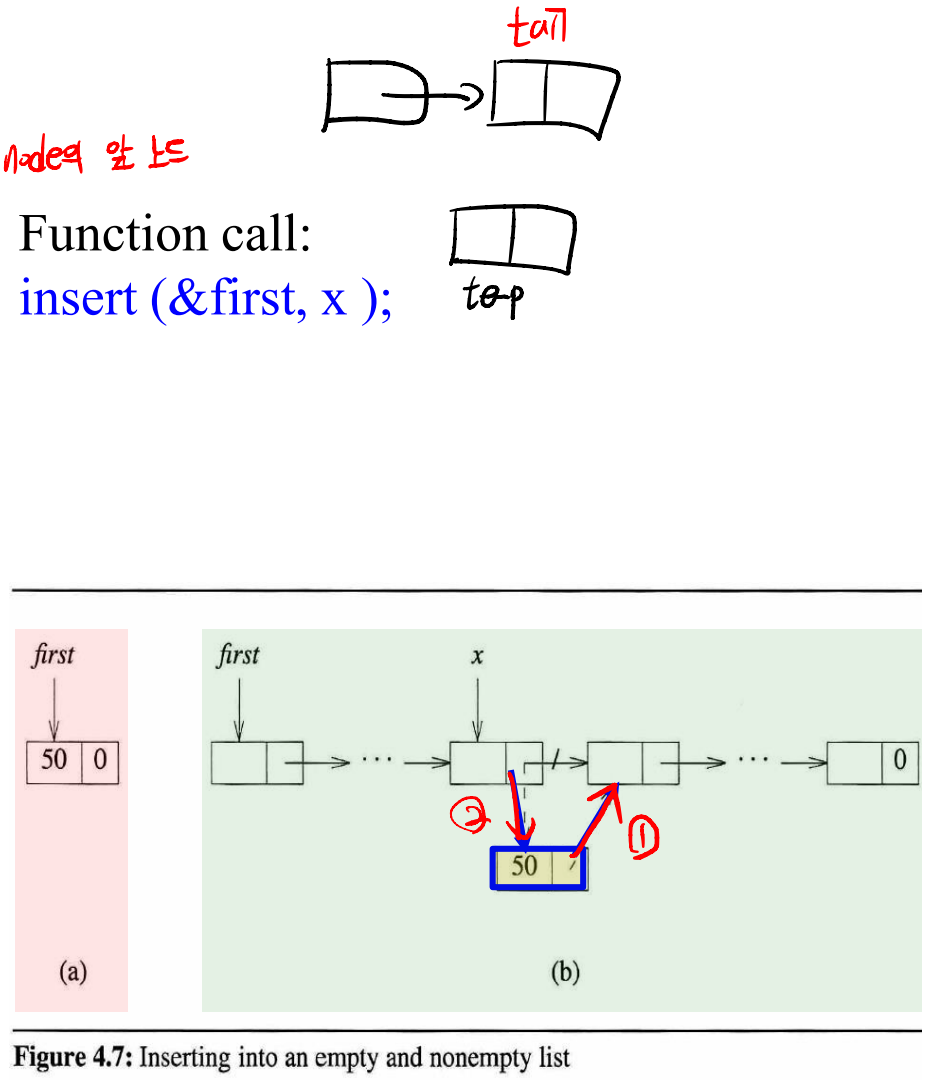
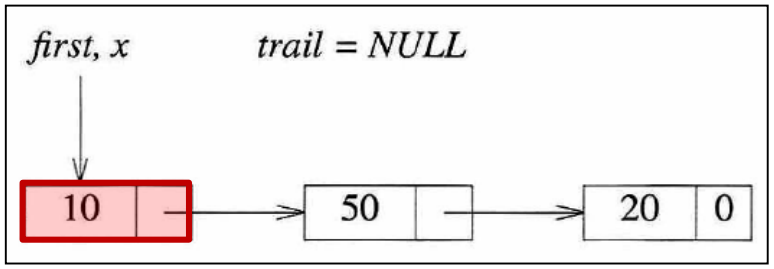
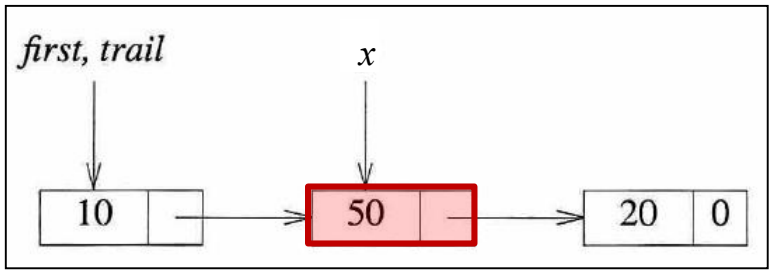



**Figure 4.7:** Inserting into an empty and nonempty list

- Ex 4.4 [List Deletion]
  - Deleting an arbitrary node from a list depends on the location of the node
    - *first* points to the start of the list
    - *x* points to the node that we wish to delete
    - *trail* points to the node that precedes *x*



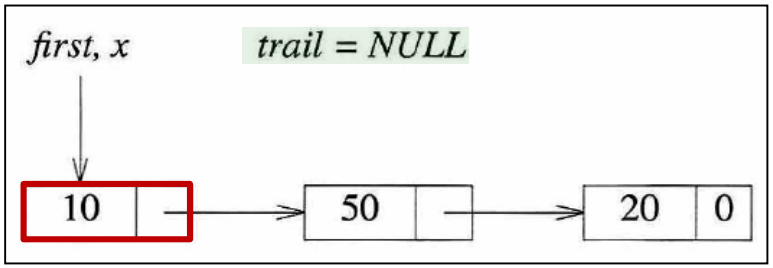

1) The *first* node

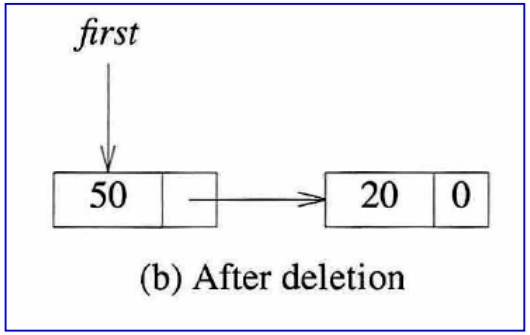



2) node other than the *first* node

1) delete the *first* node :

Function call:
delete( &first, NULL, x );

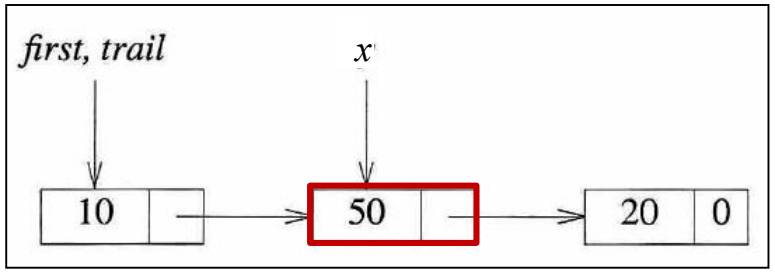


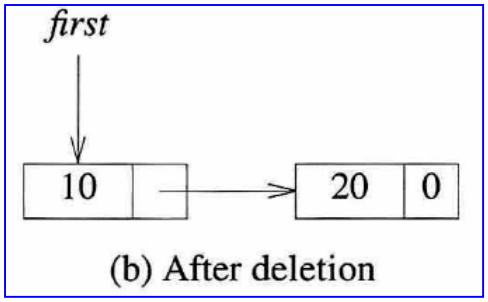




```
void delete(listPointer *first,  listPointer trail,  listPointer x )
{
     if (trail)
          trail→link = x→link;
     else
          *first = (*first)→link;
     free(x);
}
```

**Program 4.3:** Deletion from a list

2) delete node other than the *first* node : delete( &first, trail, x );

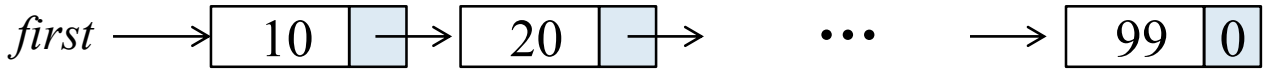



```
void delete(listPointer *first, listPointer trail, listPointer x)
{
    if (trail)
        trail→link = x→link;
    else
        *first = (*first)→link;
    free(x);
}
```

**Program 4.3:** Deletion from a list

- Ex 4.5 [Printing out a list]:




```
printList( first );
```

```
void printList(listPointer first)
{
     printf("The list contains: ");
     for (    ;   first;  first = first->link)
          printf ( "%4d", first->data);
     printf ( " \ n" );
}
```

first는 first->link가 NULL이 될 때까지...

**Program 4.4:** Printing a list

# LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.8 Doubly Linked Lists

"배열"을 이용한 stack, Queue의 구현

vs

"연결노드"를

- A linked stack and a linked queue
  - Adding/deleting a node: easy

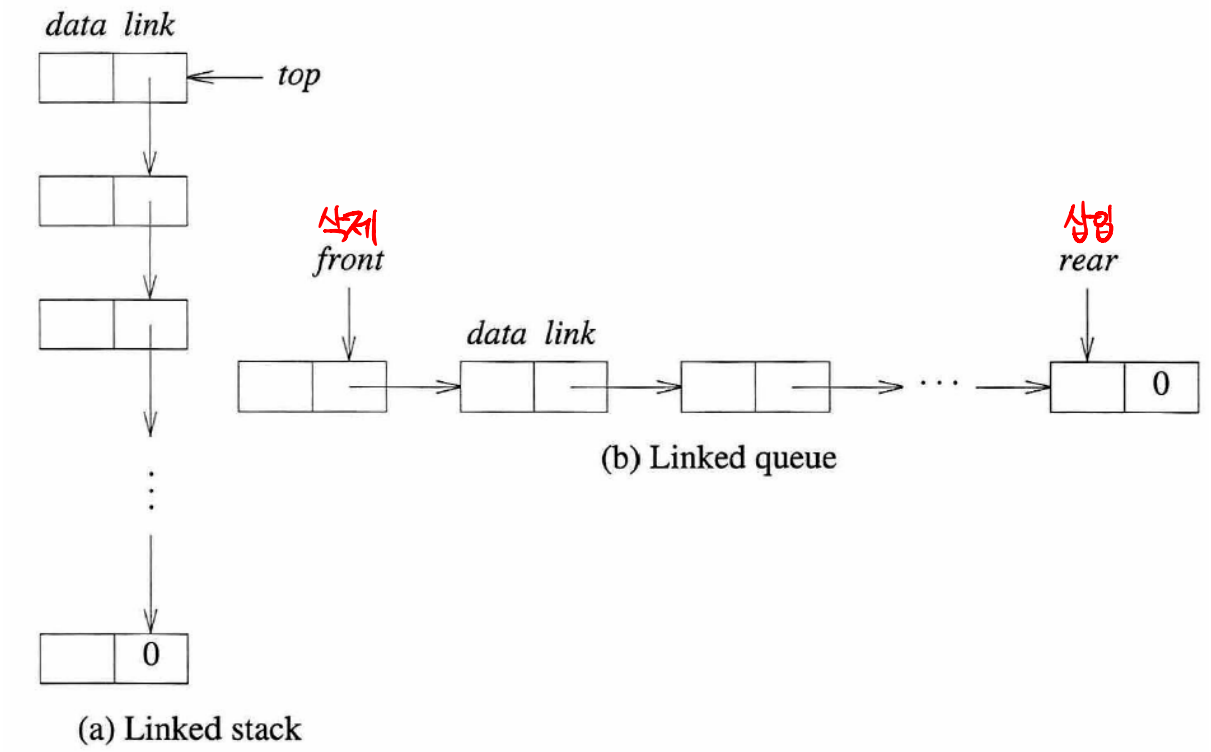
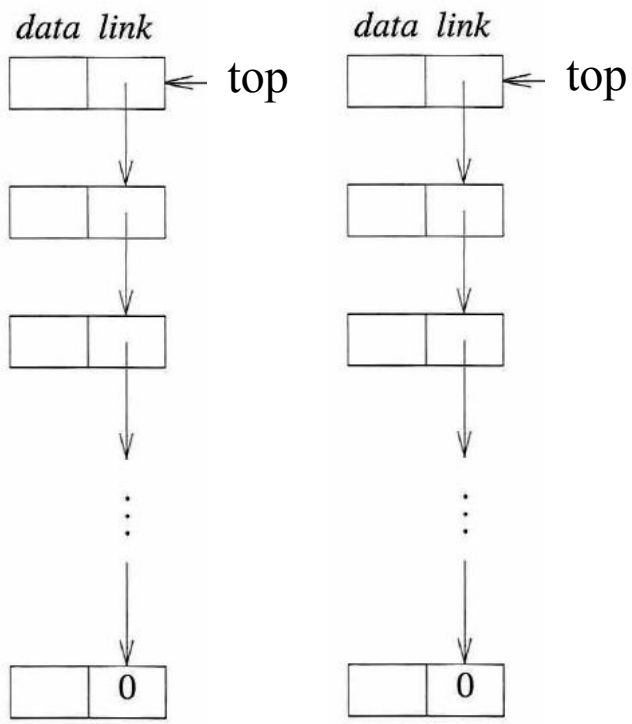



**Figure 4.11:** Linked stack and queue

여러개의 stack이 여러개 필요할때

- Represent ***n* stacks** simultaneously

```
#define MAX-STACKS 10
typedef struct {
    int key;
    /* other fields */
} element;
typedef struct stack *stackPointer;
typedef struct stack {
    element data;
    stackPointer link;
} ;
stackPointer top[MAX-STACKS];
```




  - Initial condition for the stacks
    - top[i] = NULL, $0 \le i < MAX\_STACKS$
  - Boundary condition:
    - top [i] =NULL iff the *i*th stack is empty

새로운 노드를 맨앞에 추가하는 형태

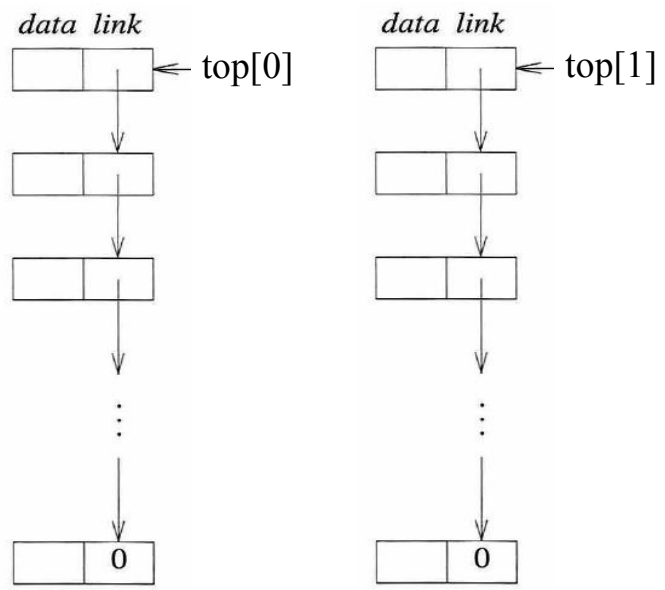
- Push in the linked stack

```
push(1, item);
```

index

```
void push(int i, element item)
{/* add item to the ith stack */
    stackPointer temp;
    MALLOC(temp, sizeof(*temp) );
    temp->data = item;
    temp->link = top[i];
    top[i] = temp;
}
```

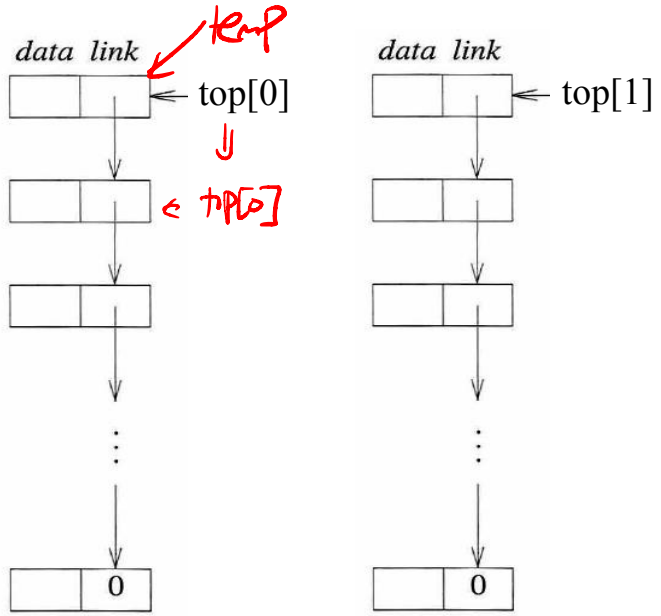
**Program 4.5:** Add to a linked stack



- Pop from the linked stack

```
item = pop(0);
```

```
element pop( int i )
{/* remove top element from the ith stack */
    stackPointer temp = top[i];
    element item;
    if (!temp)
        return stackEmpty();
    item = temp->data;
    top[i] = temp->link;
    free (temp);
    return item;
}
```

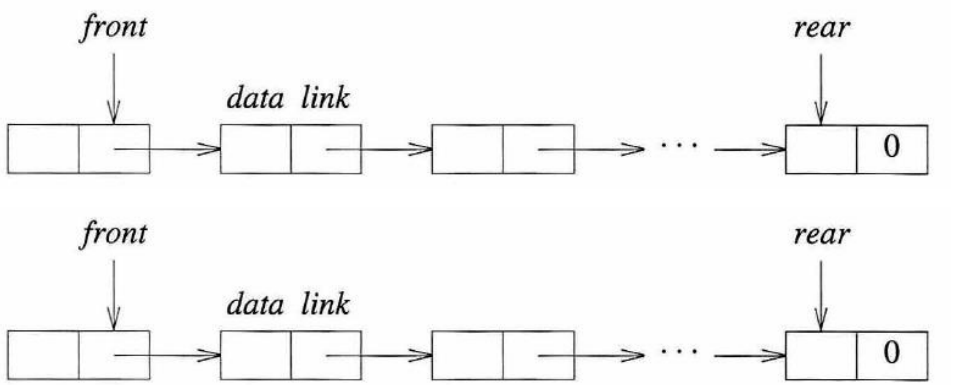



Program 4.6: Delete from a linked stack

- To represent ***m*** **queues** simultaneously

```
#define MAX-QUEUES 10 /* maximum number of queues */
typedef struct queue *queuePointer;
typedef struct queue {
   element data;
   queuePointer link;
} ;
queuePointer front[MAX-QUEUES], rear[MAX-QUEUES];
```

  - Initial condition for the queues
    - front[i] = NULL, 0≤i<MAX_QUEUES
  - Boundary condition
    - front[i] = NULL, iff the *i*th queue is empty

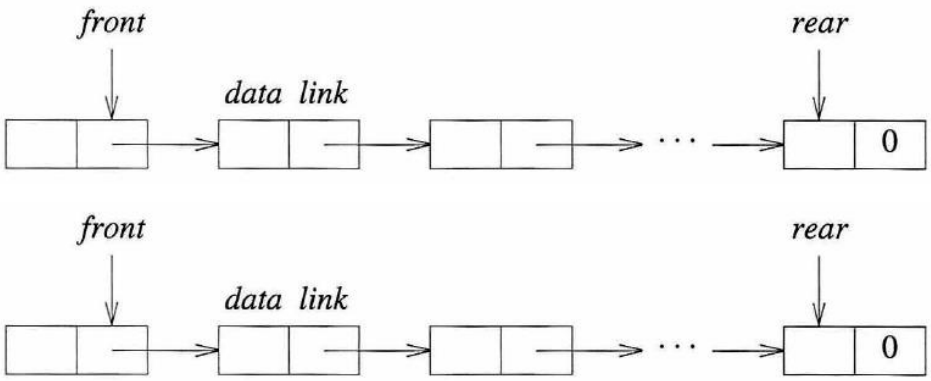


- Add operation for multiple queues

```
void addq(int i, element item)
{/* add item to the rear of queue i */
    queuePointer temp;
    MALLOC(temp, sizeof(*temp));
    temp->data = item;
    temp->link = NULL;
    if (front [i])
      rear[i]->link = temp;
    else
      front[i] = temp;
    rear[i] =temp;
}
```

```
addq(0,item);
```

**Program 4.7:** Add to the rear of a linked queue

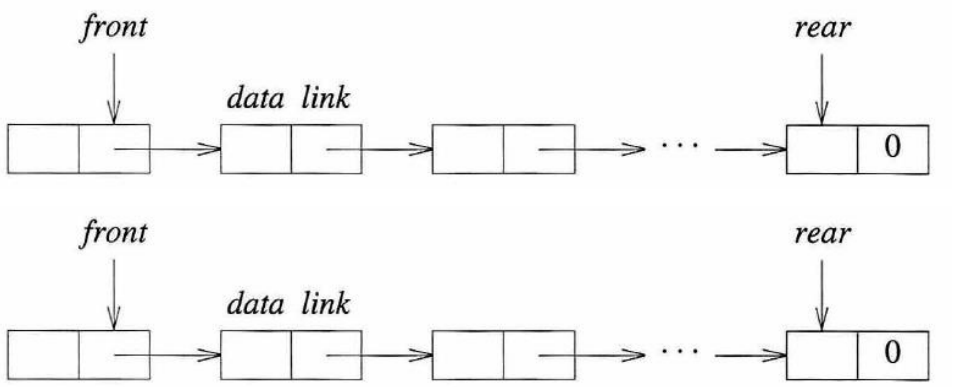


- Delete operation for multiple queues

```
element deleteq(int i)
{/* delete an element from queue i */
      queuePointer temp = front[i];
      element item;
      if (!temp)
          return queueEmpty();
      item = temp->data;
      front[i] = temp->link;
      free (temp) ;
      return item;
}
```

```
item = deleteq(i);
```

**Program 4.7:** Add to the rear of a linked queue



# LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists