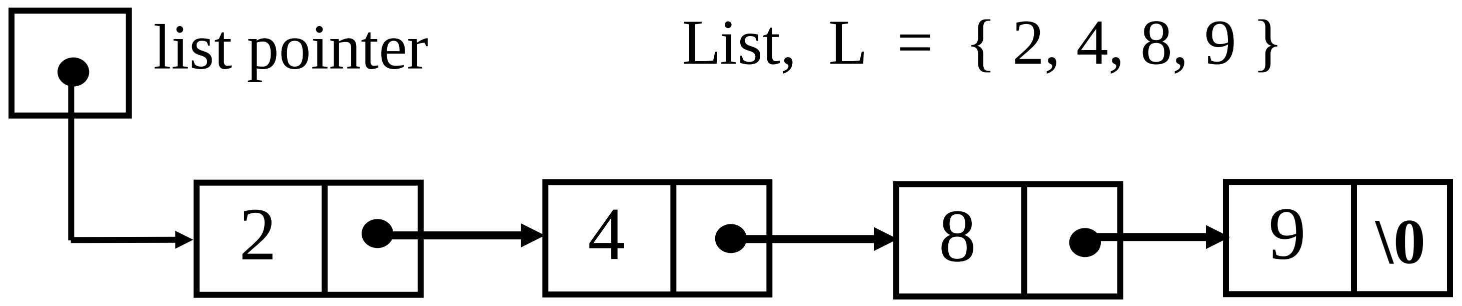
Linked List

List

- 순서가 의미를 갖는 data의 집합
- S1 = {2, 4, 8, 9}
- S2 = {“hat”, “rat”, “bat”, “cat”, “fat”}
- S3 = {4, 8, 2, 9}
- S1과 S3는 같은 집합, 서로 다른 list
- List의 구현 : array or linked list
- Array는 dynamic data에 부적합

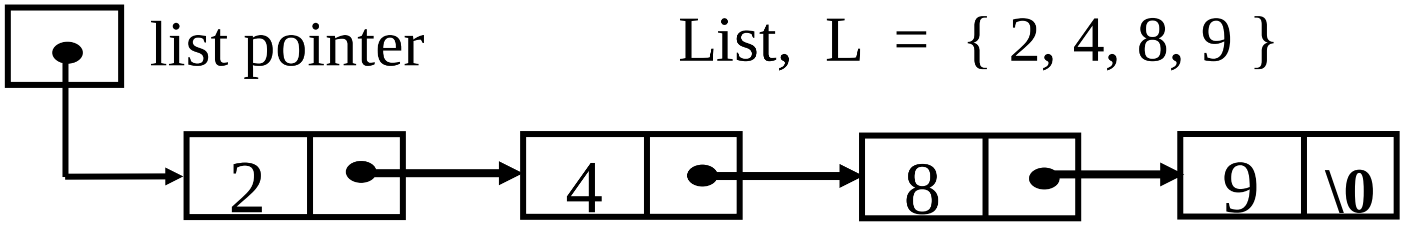
Self-Reference Data Structure

```
typedef struct list_node {
        struct list_node *link;        /* self reference */
        int data;                      /* data */
} NODE;
```



Linked List

```
typedef struct list_node {
        struct list_node *link;     /* self reference */
        int data;                   /* data */
} NODE;
```

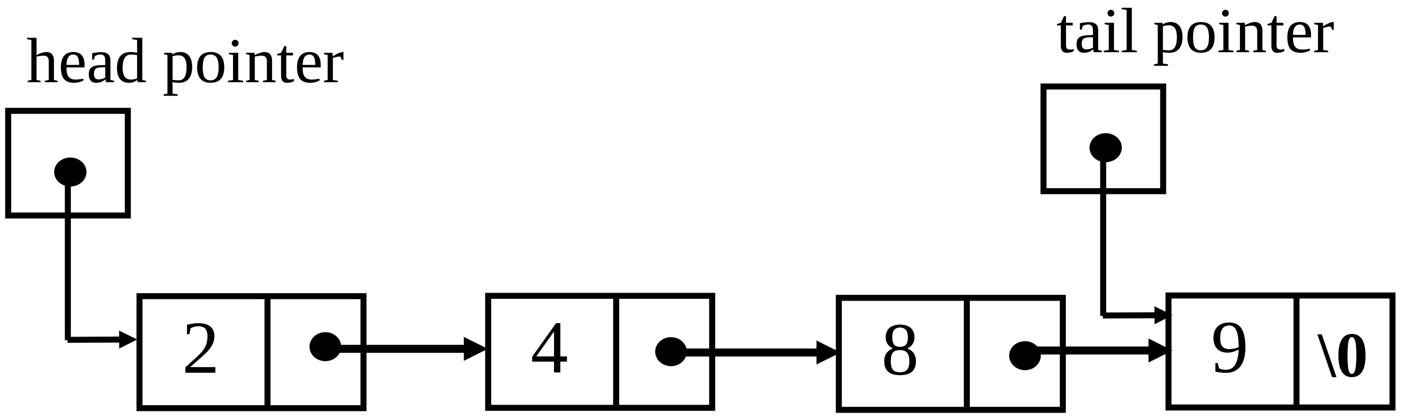


List, L = { 2, 4, 8, 9 }

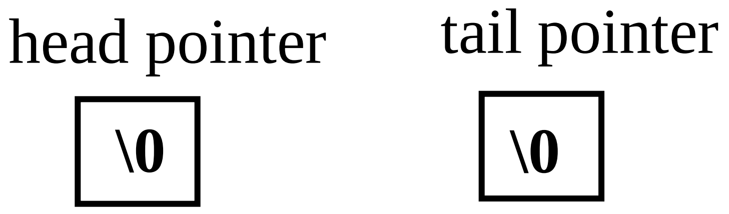
- All the boxes are of type NODE or NODE *
- Need dynamic memory allocation for the 4 boxes
  void *malloc(size_t), free(void *);
- NODE *list_pointer = 0 for empty list.

Linked List with 2 Pointers

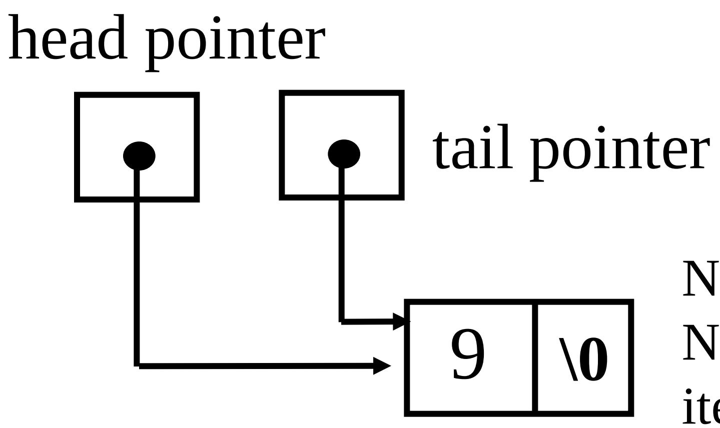
- "head pointer" and "tail pointer"



Empty List, List with 1 item



```
NODE *head = 0;
NODE *tail = 0;
```



```
NODE *head = 0;  NODE *tail = 0;
NODE *item = (NODE *)malloc(sizeof(NODE));
item->link = 0;
item->data = 9;
head = tail = item;
```

Operation on Linked List

- Insert an item to head of a list
- Delete an item from head of a list
- Insert an item to tail of a list
- Delete an item next to head item
- Delete all the items in a list
- Print data of a list

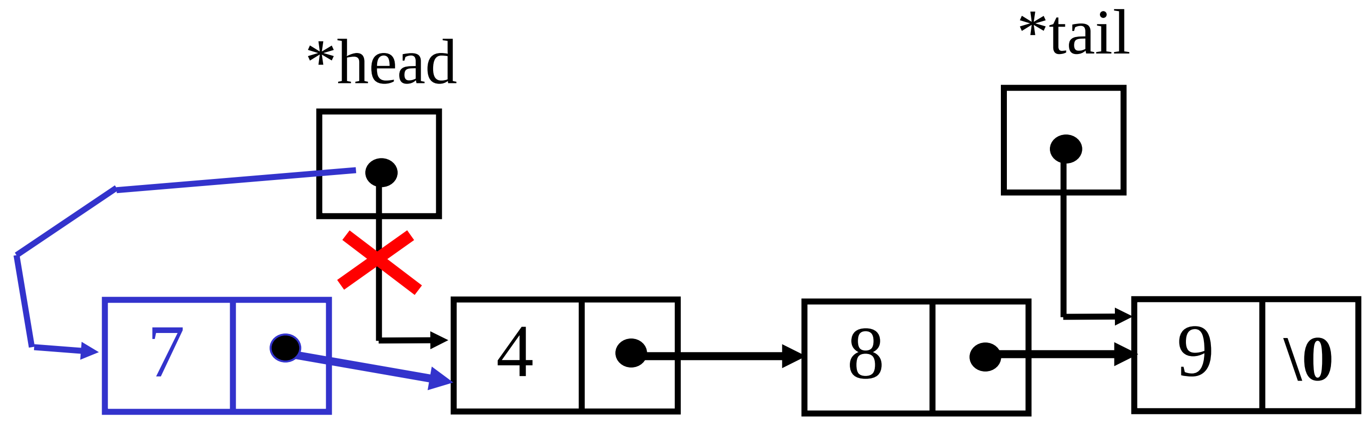
int headInsert(NODE **head, NODE **tail, int i_data)

call하기 전	call 이후	
• S1 = { }	S1 = { 7 }	i_data = 7
• S2 = { 4 }	S2 = { 7, 4 }	
• S3 = { 4, 5 }	S3 = { 7, 4, 5 }	

```
NODE *head_ptr = 0;
NODE *tail_ptr = 0;
headInsert(&head_ptr, &tail_ptr, 7);
headInsert(&head_ptr, &tail_ptr, 8);.
```

int headInsert(NODE **head, NODE **tail, int i_data)

Function returns 1 if data is inserted, 0 if fails to insert

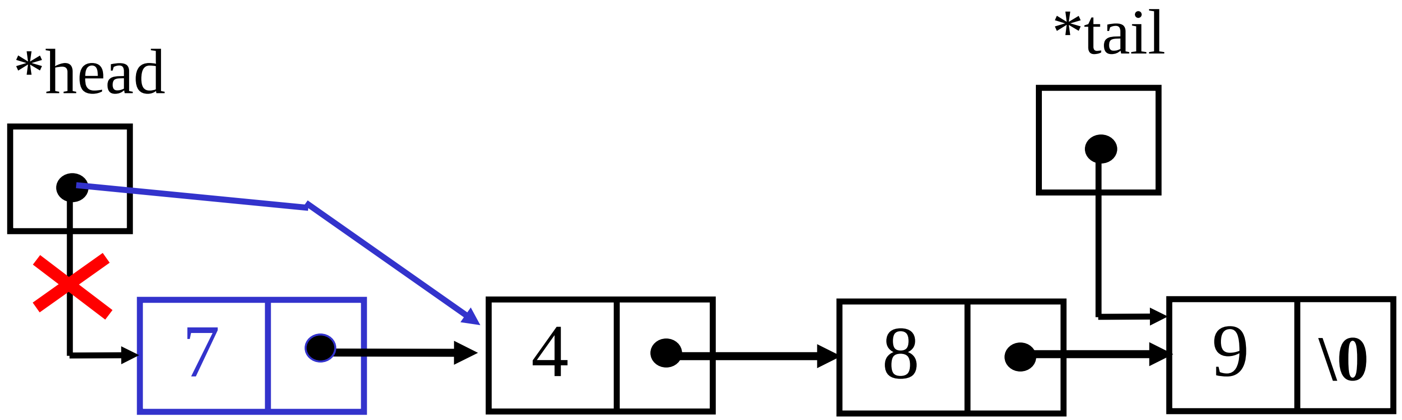


```
NODE *item = (NODE *)malloc(sizeof(NODE));          create element
if(!item) { fprintf(stderr, “out of memory\n”); return 0;}
item->data = i_data;                                fill data field
item->link = *head;                                 fill link field
*head = item;                                       connect element to head
```

```
if(!*head)
    *tail = item;
```

int headDelete(NODE **head, NODE **tail, int *i_data)

Function returns 1 if data is deleted, 0 if fails to delete

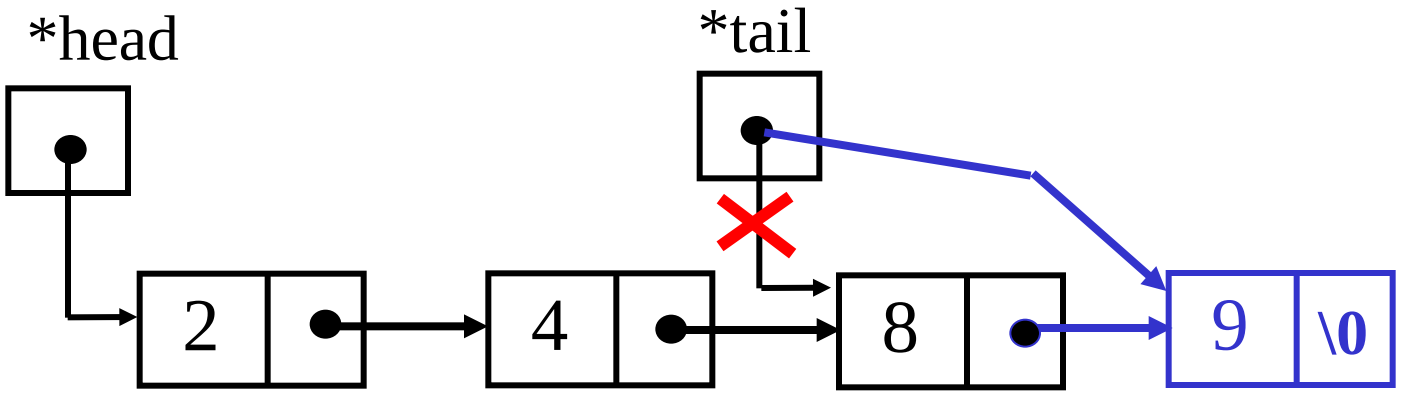


NODE *item = *head;	element to be deleted
if(!item) return 0;	empty list
*I_data = item->data;	get data value at head
*head = item->link;	head pointer to next element
free(item);	dispose memory

```
if(!*head)
    *tail = 0;
```

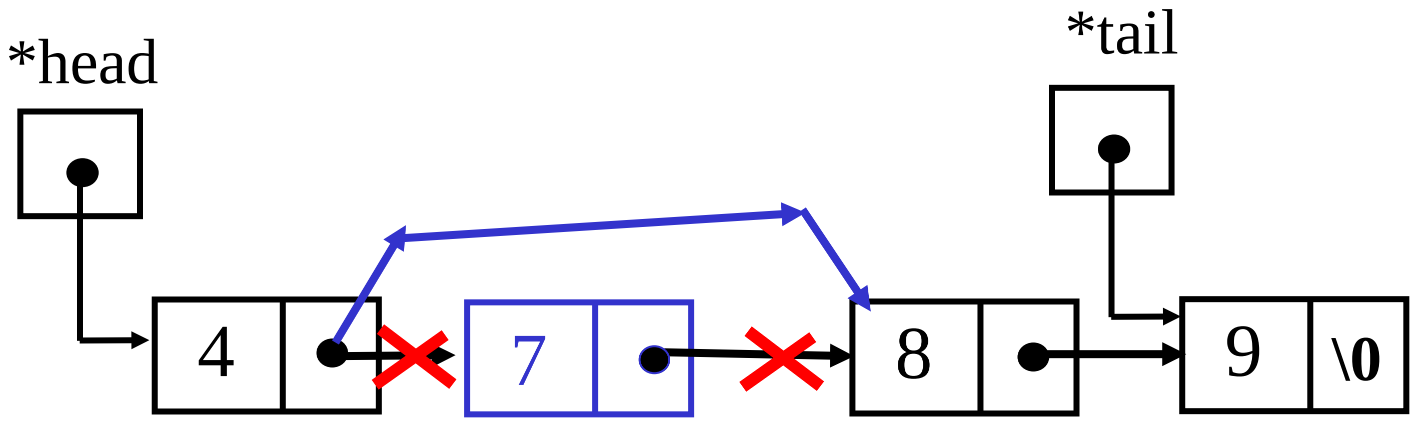
int tailInsert(NODE **head, NODE **tail, int i_data)

Function returns 1 if data is inserted, 0 if fails to insert



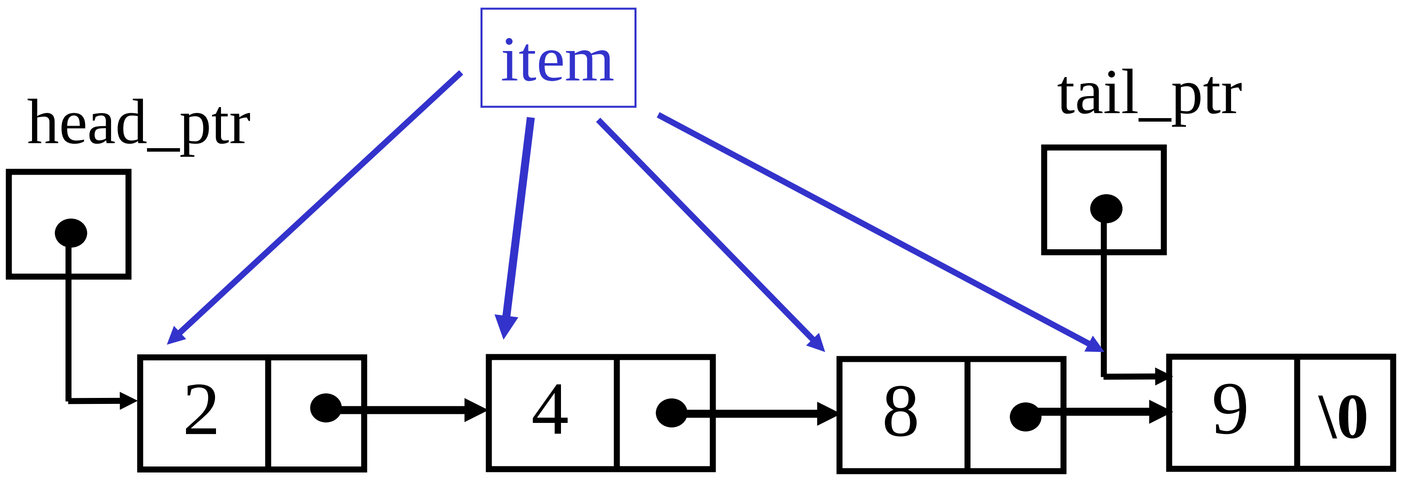
int secondDelete(NODE **head, NODE **tail, int i_data)

Function returns 1 if data is deleted, 0 if fails to delete



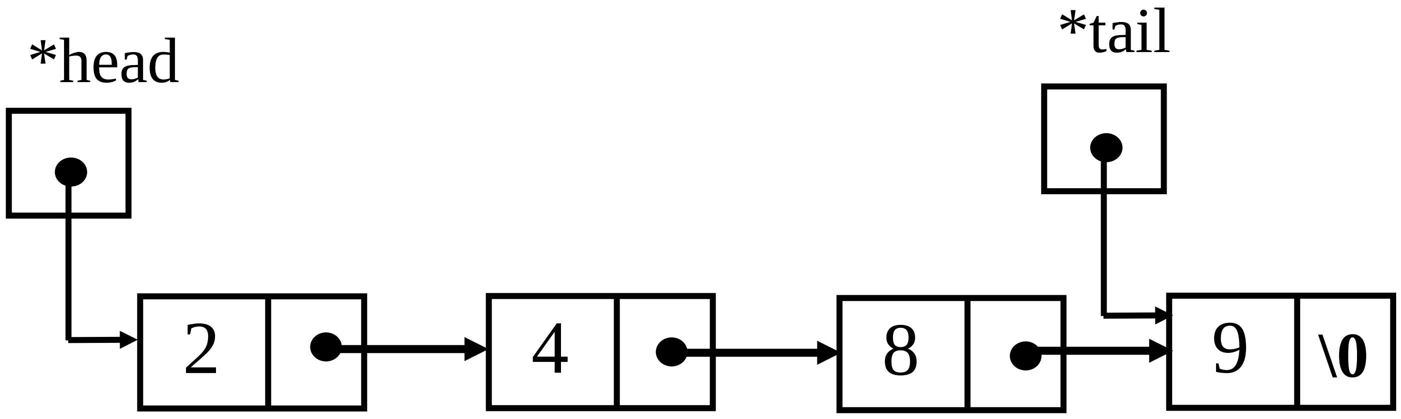
- The same as "headDelete()"
  if LIST is empty or has 1 item.
- Otherwise, 2nd item, next to head item, is deleted;

void printList(NODE *head_ptr)



```
NODE *item = head_ptr;
while(item) {
            printf(" %d", item->data);
            item = item->link;
}
```

void deleteList(NODE **head, NODE **tail)



```
NODE *item = *head;
while(item) {
        free(item);
        item = item->link;
}
*head = *tail = 0;
```

This code is wrong !!

실습

- linked_list.c is given
- void printList(NODE *) is given
- Write the other 5 functions at lab hours
- Read the program codes as a pre-lab work