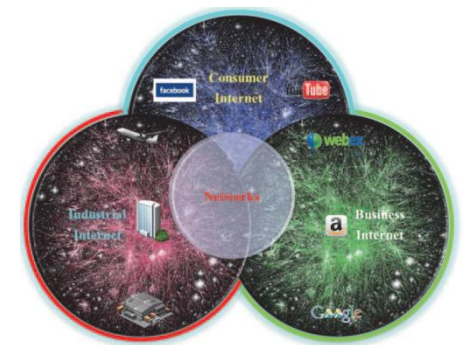
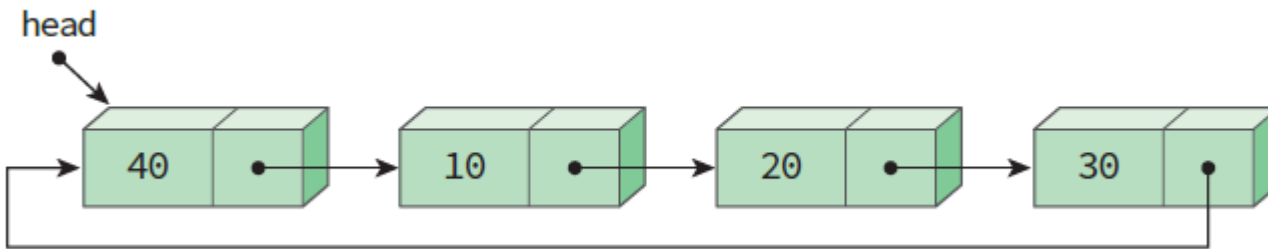


Prof. Chang Choi

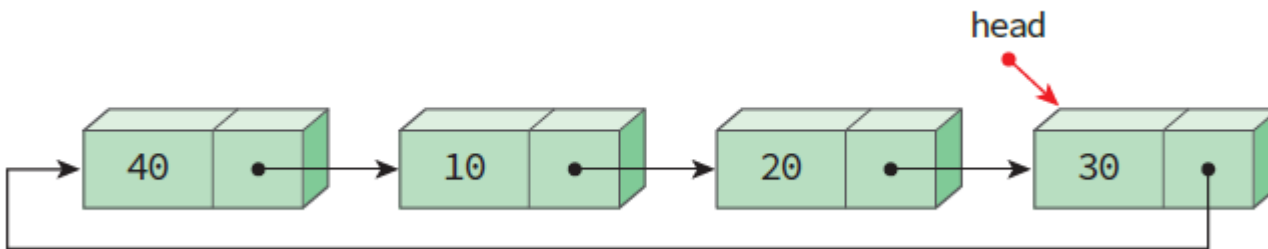


원형 연결 리스트

- 마지막 노드의 링크가 첫 번째 노드를 가리키는 리스트
- 한 노드에서 다른 모든 노드로의 접근이 가능

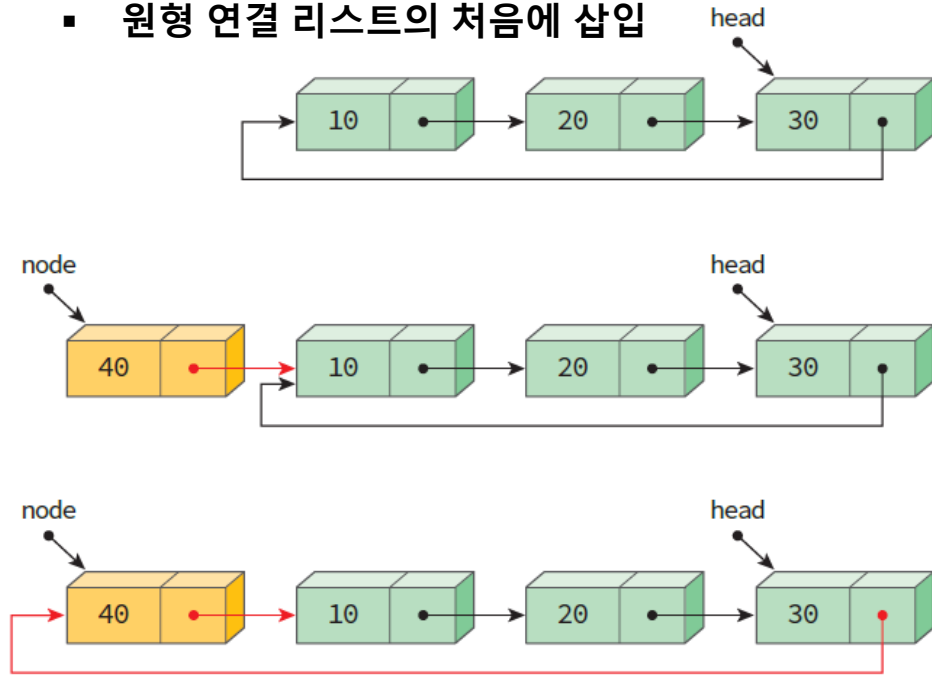


- 보통 헤드포인트가 마지막 노드를 가리키게끔 구성하면 리스트의 처음이나 마지막에 노드를 삽입하는 연산이 단순 연결 리스트에 비하여 용이

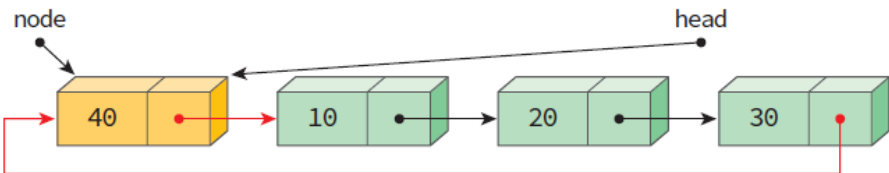


원형 연결 리스트의 삽입

■ 원형 연결 리스트의 처음에 삽입



■ 원형 연결 리스트의 끝에 삽입



```

ListNode* insert_first(ListNode* head, element data)
{
    ListNode *node = (ListNode *)malloc(sizeof(ListNode));
    node->data = data;
    if (head == NULL) {
        head = node;
        node->link = head;
    }
    else {
        node->link = head->link;    // (1)
        head->link = node;          // (2)
        head = node;                // (3)
    }
    return head;    // 변경된 헤드 포인터를 반환한다.
}
    
```

리스트를 역순으로 만드는 연산 구현

```
#include <stdio.h>
#include <stdlib.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)

typedef int element;
typedef struct ListNode { // 노드 타입
    element data;
    struct ListNode* link;
} ListNode;

// 리스트의 항목 출력
void print_list(ListNode* head)
{
    ListNode* p;

    if (head == NULL) return;
    p = head->link;
    do {
        printf("%d->", p->data);
        p = p->link;
    } while (p != head);
    printf("%d->", p->data); // 마지막 노드 출력
}
```

```
ListNode* insert_first(ListNode* head, element data)
{
    ListNode* node = (ListNode*)malloc(sizeof(ListNode));
    node->data = data;
    if (head == NULL) {
        head = node;
        node->link = head;
    }
    else {
        node->link = head->link;    // (1)
        head->link = node;        // (2)
    }
    return head; // 변경된 헤드 포인터를 반환한다.
}

ListNode* insert_last(ListNode* head, element data)
{
    ListNode* node = (ListNode*)malloc(sizeof(ListNode));
    node->data = data;
    if (head == NULL) {
        head = node;
        node->link = head;
    }
    else {
        node->link = head->link;    // (1)
        head->link = node;        // (2)
        head = node;              // (3)
    }
    return head; // 변경된 헤드 포인터를 반환한다.
}
```

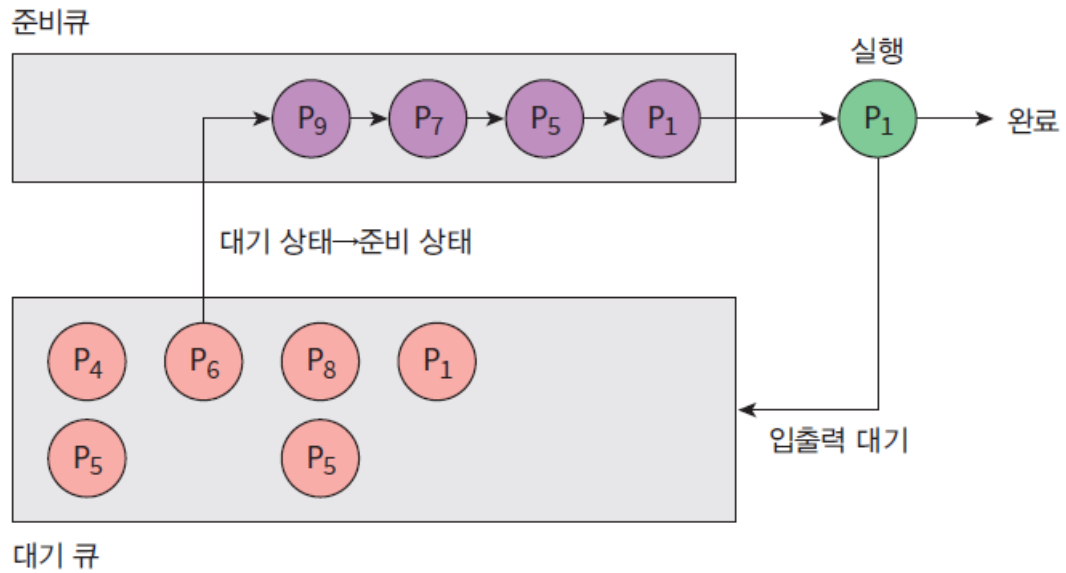
```
int main(void)
{
    ListNode* head = NULL;

    // list = 10->20->30->40
    head = insert_last(head, 20);
    head = insert_last(head, 30);
    head = insert_last(head, 40);
    head = insert_first(head, 10);
    print_list(head);
    return 0;
}
```

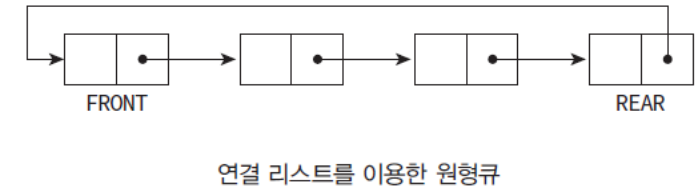
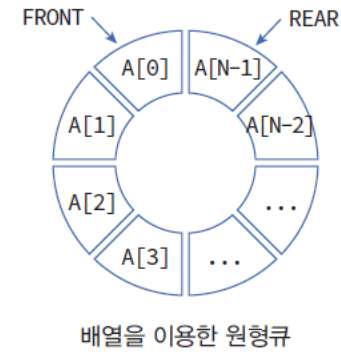
Microsoft Visual Studio
10->20->30->40->

원형 연결 리스트의 응용

컴퓨터 응용 프로그램



원형큐 구현



멀티 플레이어 게임

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)

typedef char element[100];
typedef struct ListNode { // 노드 타입
    element data;
    struct ListNode* link;
} ListNode;

typedef struct CListType {
    ListNode* head;
} CListType;
```

```
// 리스트의 항목 출력
void print_list(CListType* L)
{
    ListNode* p;

    if (L->head == NULL) return;
    p = L->head->link;
    do {
        printf("%s->", p->data);
        p = p->link;
    } while (p != L->head);
    printf("%s->", p->data); // 마지막 노드 출력
}

void insert_first(CListType* L, const element data)
{
    ListNode* node = (ListNode*)malloc(sizeof(ListNode));
    strcpy(node->data, data);
    if (L->head == NULL) {
        L->head = node;
        node->link = L->head;
    }
    else {
        node->link = L->head->link; // (1)
        L->head->link = node; // (2)
    }
}
```

```
// 원형 연결 리스트 테스트 프로그램
int main(void)
{
    CListType list = { NULL };

    insert_first(&list, "KIM");
    insert_first(&list, "PARK");
    insert_first(&list, "CHOI");

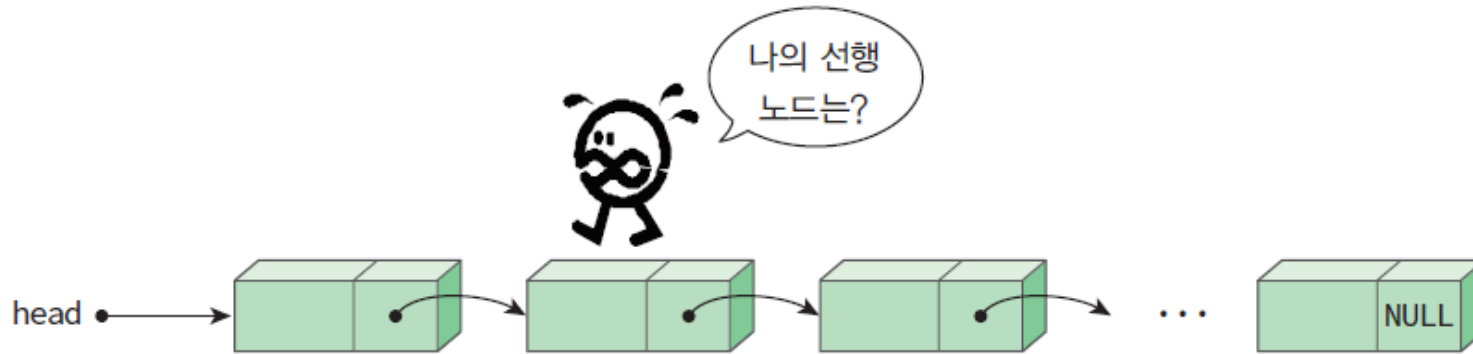
    ListNode* p = list.head;
    for (int i = 0; i < 10; i++) {
        printf("현재 차례=%s\n", p->data);
        p = p->link;
    }
    return 0;
}
```



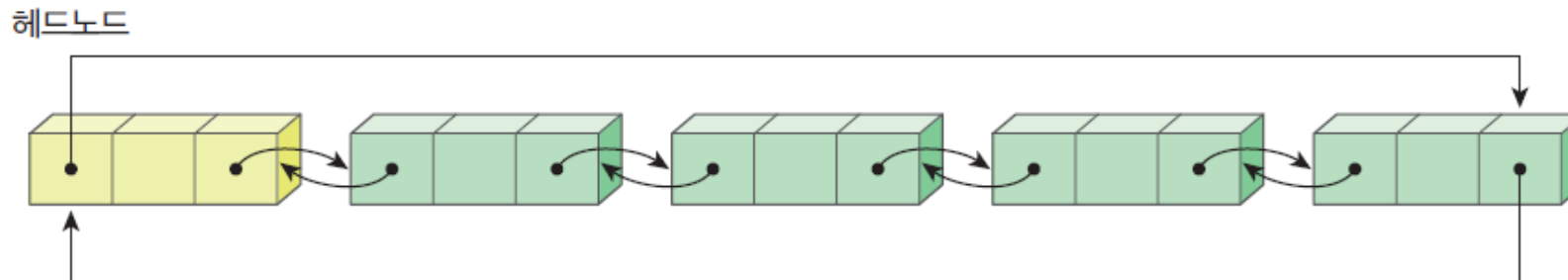
```
Microsoft Visual Studio
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
```

이중 연결 리스트

- 단순 연결 리스트의 문제점: 선행 노드를 찾기가 힘들다



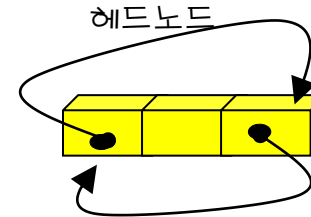
- 이중 연결 리스트: 하나의 노드가 선행 노드와 후속 노드에 대한 두 개의 링크를 가지는 리스트
- 단점은 공간을 많이 차지하고 코드가 복잡



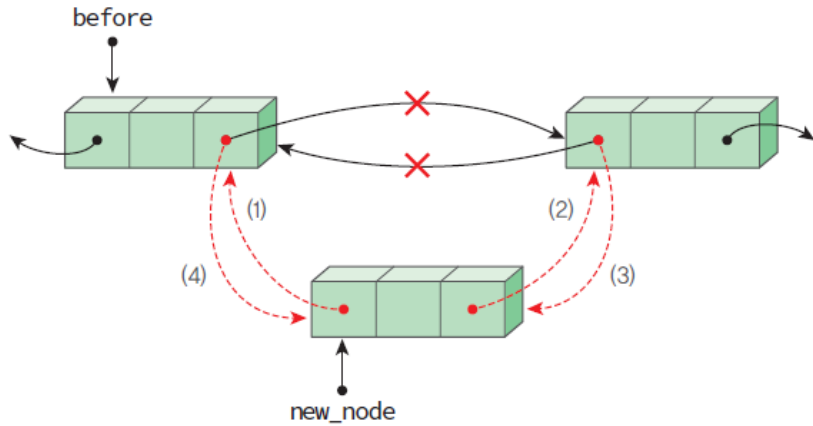
헤드 노드

- 헤드노드(head node): 데이터를 가지지 않고 단지 삽입, 삭제 코드를 간단하게 할 목적으로 만들어진 노드
 - 헤드 포인터와의 구별 필요
 - 공백상태에서는 헤드 노드만 존재

```
typedef int element;
typedef struct DlistNode {
    element data;
    struct DlistNode *llink;
    struct DlistNode *rlink;
} DlistNode;
```



삽입 및 삭제 연산



// 새로운 데이터를 노드 before의 오른쪽에 삽입한다.

```
void dinsert(DListNode *before, element data)
```

```
{
```

```
    DListNode *newnode = (DListNode *)malloc(sizeof(DListNode));
```

```
    strcpy(newnode->data, data);
```

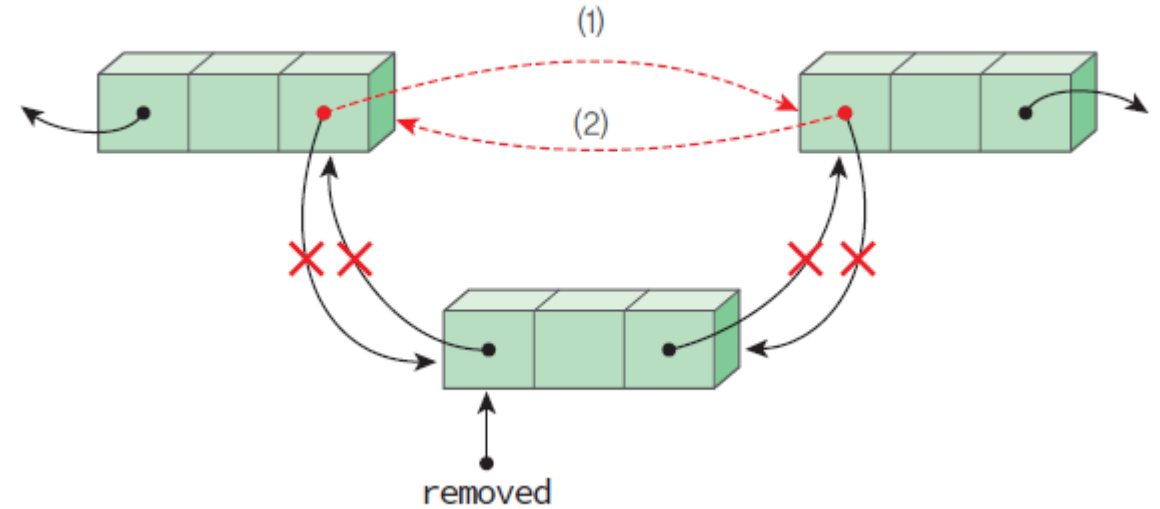
```
    newnode->llink = before;
```

```
    newnode->rlink = before->rlink;
```

```
    before->rlink->llink = newnode;
```

```
    before->rlink = newnode;
```

```
}
```



// 노드 removed를 삭제한다.

```
void ddelete(DListNode* head, DListNode* removed)
```

```
{
```

```
    if (removed == head) return;
```

```
    removed->llink->rlink = removed->rlink;
```

```
    removed->rlink->llink = removed->llink;
```

```
    free(removed);
```

```
}
```

이중 연결 리스트 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)
```

```
typedef int element;
typedef struct DListNode {
    // 이중연결 노드 타입
    element data;
    struct DListNode* llink;
    struct DListNode* rlink;
} DListNode;
```

```
// 이중 연결 리스트를 초기화
void init(DListNode* phead)
{
    phead->llink = phead;
    phead->rlink = phead;
}
```

```
// 이중 연결 리스트의 노드를 출력
void print_dlist(DListNode* phead)
{
    DListNode* p;
    for (p = phead->rlink; p != phead; p = p->rlink) {
        printf("<-| %d| -> ", p->data);
    }
    printf("\n");
}
```

// 새로운 데이터를 노드 before의 오른쪽에 삽입한다.

```
void dinsert(DListNode* before, element data)
{
    DListNode* newnode = (DListNode*)malloc(sizeof(DListNode));
    newnode->data = data;
    newnode->llink = before;
    newnode->rlink = before->rlink;
    before->rlink->llink = newnode;
    before->rlink = newnode;
}
```

// 노드 removed를 삭제한다.

```
void ddelete(DListNode* head, DListNode* removed)
{
    if (removed == head) return;
    removed->llink->rlink = removed->rlink;
    removed->rlink->llink = removed->llink;
    free(removed);
}
```

// 이중 연결 리스트 테스트 프로그램

```
int main(void)
{
    DListNode* head = (DListNode*)malloc(sizeof(DListNode));
    init(head);
    printf("추가 단계\n");
    for (int i = 0; i < 5; i++) {
        // 헤드 노드의 오른쪽에 삽입
        dinsert(head, i);
        print_dlist(head);
    }
    printf("\n삭제 단계\n");
    for (int i = 0; i < 5; i++) {
        print_dlist(head);
        ddelete(head, head->rlink);
    }
    free(head);
    return 0;
}
```

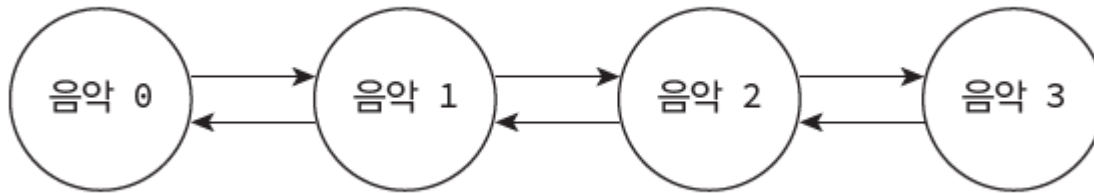
```
Microsoft Visual Studio 디버그 콘솔

추가 단계
<-| 0| ->
<-| 1| -> <-| 0| ->
<-| 2| -> <-| 1| -> <-| 0| ->
<-| 3| -> <-| 2| -> <-| 1| -> <-| 0| ->
<-| 4| -> <-| 3| -> <-| 2| -> <-| 1| -> <-| 0| ->

삭제 단계
<-| 4| -> <-| 3| -> <-| 2| -> <-| 1| -> <-| 0| ->
<-| 3| -> <-| 2| -> <-| 1| -> <-| 0| ->
<-| 2| -> <-| 1| -> <-| 0| ->
<-| 1| -> <-| 0| ->
<-| 0| ->
```

MP3 재생 프로그램

- [Ch 7] mp3_play.c파일 참조

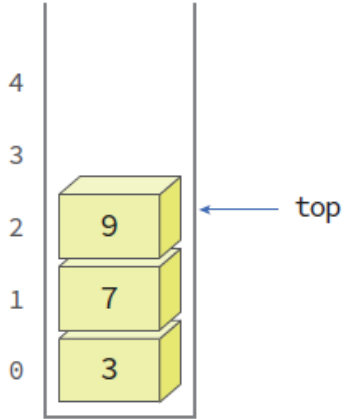


Microsoft Visual Studio 디버그 콘솔

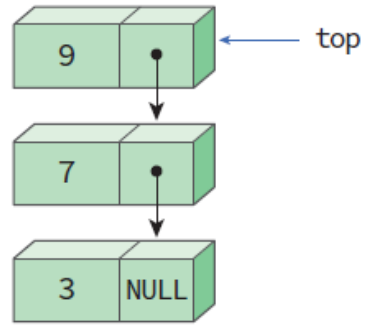
```

<-| #Fernando# |-> <-| Dancing Queen |-> <-| Mamamia |->
명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->
명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| Dancing Queen |-> <-| #Mamamia# |->
명령어를 입력하시오(<, >, q): <
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->
명령어를 입력하시오(<, >, q): q
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->
  
```

연결 리스트로 구현한 스택



(a) 배열을 이용한 스택

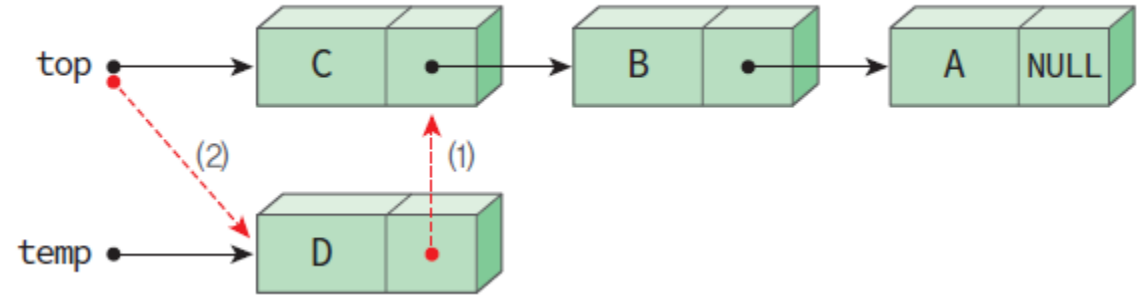


(b) 연결 리스트를 이용한 스택

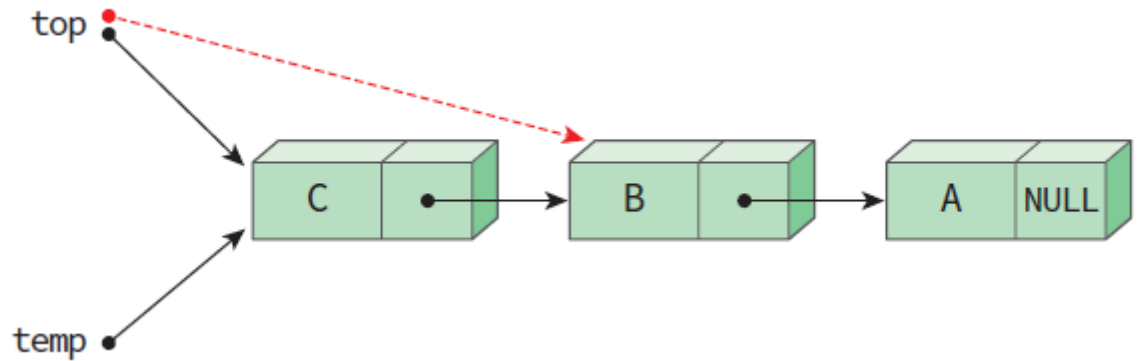
```
typedef int element;
typedef struct StackNode {
    element data;
    struct StackNode *link;
} StackNode;
```

```
typedef struct {
    StackNode *top;
} LinkedStackType;
```

삽입 연산



삭제 연산



연결 리스트 스택 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)
```

```
typedef int element;
typedef struct StackNode {
    element data;
    struct StackNode* link;
} StackNode;
```

```
typedef struct {
    StackNode* top;
} LinkedStackType;
```

```
// 초기화 함수
void init(LinkedStackType* s)
{
    s->top = NULL;
}
```

```
// 공백 상태 검출 함수
int is_empty(LinkedStackType* s)
{
    return (s->top == NULL);
}
```

```
// 포화 상태 검출 함수
int is_full(LinkedStackType* s)
{
    return 0;
}
```

```
// 삽입 함수
void push(LinkedStackType* s, element item)
{
    StackNode* temp = (StackNode*)malloc(sizeof(StackNode));
    temp->data = item;
    temp->link = s->top;
    s->top = temp;
}
```

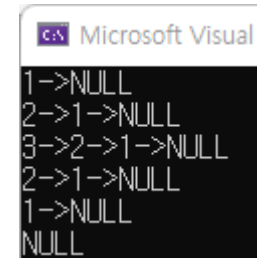
```
void print_stack(LinkedStackType* s)
{
    for (StackNode* p = s->top; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL \n");
}
```

```
// 삭제 함수
element pop(LinkedStackType* s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택이 비어있음\n");
        exit(1);
    }
    else {
        StackNode* temp = s->top;
        int data = temp->data;
        s->top = s->top->link;
        free(temp);
        return data;
    }
}
```

```
// 피크 함수
element peek(LinkedStackType* s)
{
    if (is_empty(s)) {
        fprintf(stderr, "스택이 비어있음\n");
        exit(1);
    }
    else {
        return s->top->data;
    }
}
```

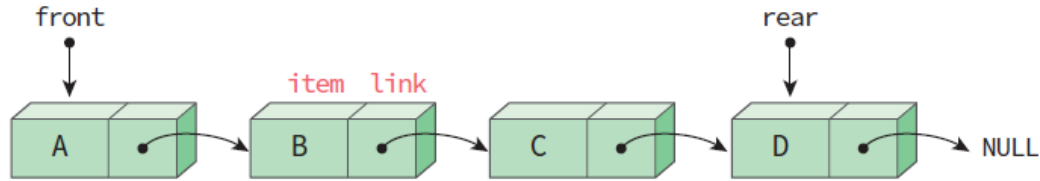
```
// 주 함수
int main(void)
{
```

```
    LinkedStackType s;
    init(&s);
    push(&s, 1); print_stack(&s);
    push(&s, 2); print_stack(&s);
    push(&s, 3); print_stack(&s);
    pop(&s); print_stack(&s);
    pop(&s); print_stack(&s);
    pop(&s); print_stack(&s);
    return 0;
}
```



```
Microsoft Visual C++
1->NULL
2->1->NULL
3->2->1->NULL
2->1->NULL
1->NULL
NULL
```

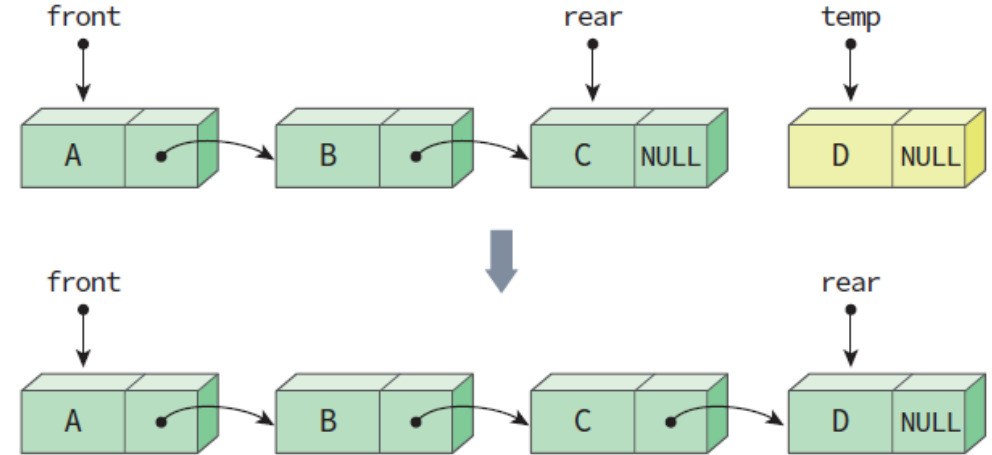
연결 리스트로 구현한 큐



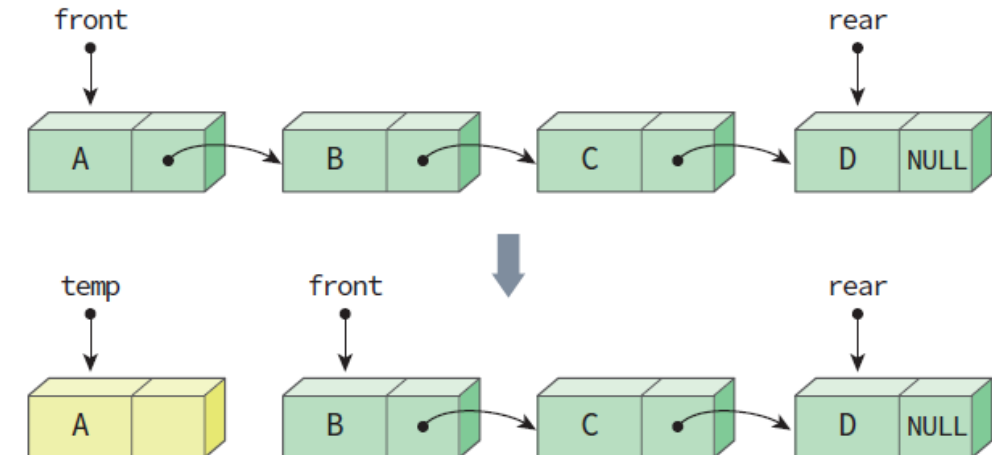
```
typedef int element;           // 요소의 타입
typedef struct QueueNode {     // 큐의 노드의 타입
    element data;
    struct QueueNode *link;
} QueueNode;

typedef struct {               // 큐 ADT 구현
    QueueNode *front, *rear;
} LinkedQueueType;
```

삽입 연산



삭제 연산



연결 리스트 큐 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable:4996)

typedef int element; // 요소의 타입
typedef struct QueueNode { // 큐의 노드의 타입
    element data;
    struct QueueNode* link;
} QueueNode;

typedef struct { // 큐 ADT 구현
    QueueNode* front, * rear;
} LinkedListType;

// 큐 초기화 함수
void init(LinkedListType* q)
{ q->front = q->rear = 0; }

// 공백 상태 검출 함수
int is_empty(LinkedListType* q)
{ return (q->front == NULL); }

// 포화 상태 검출 함수
int is_full(LinkedListType* q)
{ return 0; }
```

```
// 삽입 함수
void enqueue(LinkedListType* q, element data)
{
    QueueNode* temp = (QueueNode*)malloc(sizeof(QueueNode));
    temp->data = data; // 데이터 저장
    temp->link = NULL; // 링크 필드를 NULL
    if (is_empty(q)) { // 큐가 공백이면
        q->front = temp;
        q->rear = temp;
    }
    else { // 큐가 공백이 아니면
        q->rear->link = temp; // 순서가 중요
        q->rear = temp;
    }
}

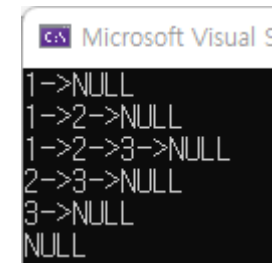
// 삭제 함수
element dequeue(LinkedListType* q)
{
    QueueNode* temp = q->front;
    element data;
    if (is_empty(q)) { // 공백상태
        fprintf(stderr, "스택이 비어있음\n");
        exit(1);
    }
    else {
        data = temp->data; // 데이터를 꺼낸다.
        q->front = q->front->link; // front를 다음노드를 가리키도록 한다.
        if (q->front == NULL) // 공백 상태
            q->rear = NULL;
        free(temp); // 동적메모리 해제
        return data; // 데이터 반환
    }
}
```

```
void print_queue(LinkedListType* q)
{
    QueueNode* p;
    for (p = q->front; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL\n");
}

// 연결된 큐 테스트 함수
int main(void)
{
    LinkedListType queue;

    init(&queue); // 큐 초기화

    enqueue(&queue, 1); print_queue(&queue);
    enqueue(&queue, 2); print_queue(&queue);
    enqueue(&queue, 3); print_queue(&queue);
    dequeue(&queue); print_queue(&queue);
    dequeue(&queue); print_queue(&queue);
    dequeue(&queue); print_queue(&queue);
    return 0;
}
```



```
Microsoft Visual S
1->NULL
1->2->NULL
1->2->3->NULL
2->3->NULL
3->NULL
NULL
```

Thank you

Q & A