데구 7장 연결리스트2

상태	완료
데드라인	@April 16, 2025
PROCESS	💚 데이터구조

▼ 원형 연결 리스트

- 마지막 노드의 링크가 첫번째 노드를 가리키는 리스트
- 한 노드에서 다른 노드로의 접근이 가능함

▼ 원형 연결 리스트 변형

- 보통 헤드 포인터가 마지막 노드를 가리키게끔 구성함
  - 리스트의 첫번째 노드는 head → link로 지정
  - 장점 : 리스트의 처음이나 마지막에 노드를 삽입하는 연산이 단순 연결 리스트에 비하여 용이

```
ListNode *insert_first(ListNode *head, e
    ListNode *node = (ListNode *)malloc(
    node→data = data;
    if(head == NULL){
        head = node;
        node→link = head;
    }else{
        node→link = head→link;
        head→link = node;
    }
    return head;
}
```

```
ListNode *insert_last(ListNode *head, e
    ListNode *node = (ListNode *)malloc(
    node→data = data;
    if(head == NULL){
        head = node;
        node→link = head;
    }else{
        node→link = head→link;
        head→link = node;
        head = node;
    }
    return head;
}
```

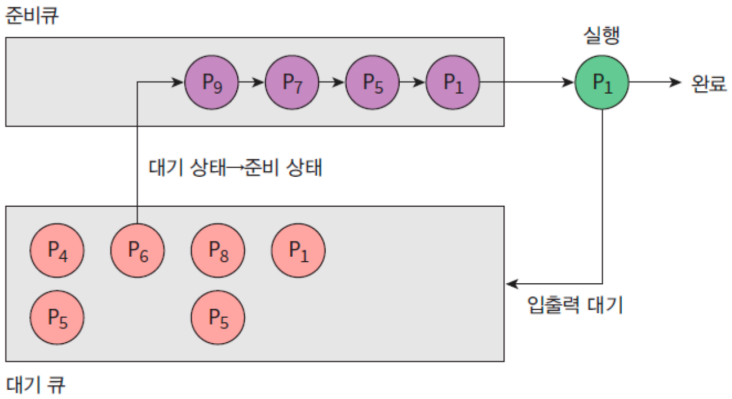
```
#include <stdio.h>
#include <stdlib.h>

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

```
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);
}
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;
        head->link = node;
    }
    return head;
}
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;
        head->link = node;
        head = node;
    }
    return head;
}

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

```
    head = insert_last(head, 20);

    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;
}
```

▼ 원형 연결 리스트 응용

- Process Scheduling



▼ 멀티플레이어 게임

head를 REAR로 head→link로 FORNT를 포인팅

insert_last(head, data)로 enqueue(q, element) 구현

delete_first(head)로 dequeue(q) 구현

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef char element[100];
typedef struct ListNode{
    element data;
    struct ListNode *link;
```

```
}ListNode;

ListNode *insert_first(ListNode *head, element data){
    ListNode *node = (ListNode *)malloc(sizeof(ListNode));
    strcpy(node→data, data);
    if(head == NULL){
        head = node;
        node→link = head;
    }else{
        node→link = head→link;
        head→link = node;
    }
    return head;
}

int main(){
    ListNode *head = NULL, *p;

    head = insert_first(head, "KIM");
    head = insert_first(head, "PARK");
    head = insert_first(head, "CHOI");

    p = head→link;
    for(int i = 0; i < 10; i++){
        printf("현재 차례 : %s\n", p→data);
        p = p→link;
    }
    return 0;
}
```

▼ 이중 연결리스트

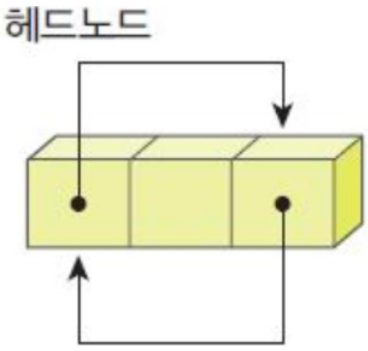
- 단순 연결리스트의 문제점 : 선행 노드를 찾기가 힘들다
- 이중 연결 리스트 : 하나의 노드가 선행 노드와 후속 노드에 대한 두 개의 링크를 가지는 리스트
- 단점 - 공간을 많이 차지하고 코드가 복잡함
- 이중 연결 리스트 + 원형 연결 리스트

▼ 헤드 노드

데이터를 가지고 있지 않고 단지 삽입, 삭제 코드를 간단하게 할 목적으로 만들어진 노드

- 일반 노드와 동일한 구조지만 데이터를 포함하지 않음 : 헤드 포인터와의 구별 필요

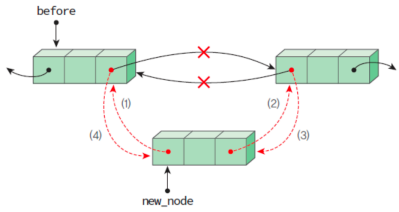
[그림 7-7] 이중 연결 리스트에서의 노드의 구조



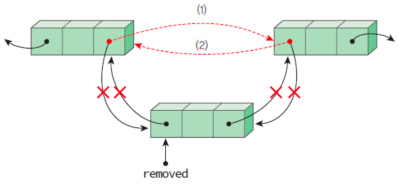
[그림 7-8] 공백상태

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

```
//삽입 연산
void dinsert(DListNode *before, element data){
    DListNode *newnode = (DListNode *)malloc(sizeof
    newnode→data = data;
    newnode→llink = before;
    newnode→rlink = before→rlink;
    before→rlink→llink = newnode;
    before→rlink = newnode;
}
```




```
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

```
#include <stdio.h>
#include <stdlib.h>

typedef int element;
typedef struct DListNode{
   element data;
   struct DListNode *llink;
   struct DListNode *rlink;
}DListNode;

void init(DListNode *phead){
   phead→llink = phead;
   phead→rlink = phead;
}

void print_dlist(DListNode *phead){
   DListNode *p;
   for(p = phead→rlink; p!=phead;p = p→rlink)
      printf("← |%d| → ", p→data);
   printf("\n");
}
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;
}

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

int main(){
   DListNode *head = (DListNode *)malloc(sizeof(DListNode));
```

```
    init(head);
    printf("추가 단계\n");
    for(int i = 0; i < 5; i++){
        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;
}
```

▼ mp3 재생 프로그램 만들기

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef char element[100];
typedef struct DListNode{
    element data;
    struct DListNode *llink;
    struct DListNode *rlink;
}DListNode;

DListNode *current;

void init(DListNode *phead){
    phead→llink = phead;
    phead→rlink = phead;
}

void print_dlist(DListNode *phead){
    DListNode *p;
    for(p = phead→rlink; p!= phead; p = p→rlink){
        if(p == current)
            printf("← | |#%s#| | →", p→data);
        else
```

```
        printf("←| |%s| |→", p→data);
    }
}

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;
}

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

int main(){
    char ch;
    DListNode *head = (DListNode *)malloc(sizeof(DListNode));
    init(head);

    dinsert(head, "Mamamia");
    dinsert(head, "Dancing Queen");
    dinsert(head, "Fernando");

    current = head→rlink;
    print_dlist(head);

    do{
        printf("\n 명령어를 입력하세요 < > q : ");
        ch = getchar();
        if(ch == '<'){
            current = current→llink;
            if(current == head)
                current = current→llink;
        }else if(ch == '>'){
            current = current→rlink;
```

```
            if(current == head)
                current = current→rlink;
        }
        print_dlist(head);
        getchar();
    }while(ch != 'q');
    free(head);
    return 0;
}
```

▼ 연결리스트로 구현한 스택

- 스택의 크기를 동적으로 결정, 크기에 대한 제한이 없고 배열과 같이 스택 크기를 매우 크게 잡을 필요가 없다
- 반면, 동적 메모리 할당이나 링크에 대한 처리로 인해 연산의 시간은 배열에 비해 더 걸릴 수 있음
- 연결리스트 구현 스택의 ADT는 배열로 구현한 스택과 완전히 동일함
  - 내부 연산 구현 등은 달라짐
  - top은 정수가 아닌 포인터로 구현함 ㅇㅇ

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

typedef struct{
    StackNode *top; //기존 배열 스택과 동일한 형식을 지원하기 위함 LinkedStackType s;
    //s→type = NULL;
}LinkedStackType;
```

▼ 연결리스트로 구현한 큐

장점 : 크기 제한이 없고, 배열처럼 크게 공간을 잡을 필요가 없음

단점 : 링크 필드에 의한 공간 낭비와 구현이 조금 복잡해짐