데구 6장 연결리스트1

상태	완료
데드라인	@April 16, 2025
PROCESS	💚 데이터구조

▼ 리스트 추상 데이터 타입

- 리스트 : 순서에 다라 연속적으로 동일한 형태의 데이터들이 저장된 공간
- 스택, 큐도 리스트의 일종임
- 삽입 연산 : 리스트에 새 항목 추가
- 삭제 연산 : 리스트에 항목 삭제
- 탐색 연산 : 리스트에서 특정 항목을 찾음

▼ ADT

- insert(list, pos, item) : pos 위치에 요소를 추가함
- insert_last(list, item) : 맨 끝에 요소를 추가함
- insert_first(list, item) : 맨 처음에 요소를 추가함
- delete(list, pos) : pos 위치에 요소를 제거함
- clear(list) : 리스트의 모든 요소를 제거함
- get_entry(list, pos) : pos위치의 요소를 반환함
- get_length(list) : 리스트의 길이를 구함
- is_empty(list) : 리스트가 비었냐
- is_full(list) : 리스트가 찼냐
- print_list : 리스트의 모든 요소를 표시함

▼ 배열로 구현된 리스트

- 배열을 이용하므로 리스트를 구현하면 순차적인 메모리 공간이 할당되므로 이것을 리스트의 순차적 표현 sequential representation이라고 함

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

#define MAX 100
```

```
typedef int element;
typedef struct {
    element array[MAX];
    int size;
}ArrayListType;

void error(const char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}
void init(ArrayListType *L){
    L → size = 0;
}
int is_empty(ArrayListType *L){
    return L→size == 0;
}

int is_full(ArrayListType *L){
    return L→size == MAX;
}
element get_entry(ArrayListType *L, int pos){
    if(pos < 0 || pos >= L→size)
        error("위치 오류");
    return L→array[pos];
}

void insert_last(ArrayListType *L, element item){
    if(L→size == MAX)
        error("리스트 오버플로우");
    L→array[L→size++] = item;
}
void insert(ArrayListType *L, int pos, element item){
    if(!is_full(L) && (pos >= 0) && (pos <= L→size)){
        for(int i = (L→size -1); i >= pos; i--)
            L→array[i+1] = L→array[i];
        L→array[pos] = item;
        L→size++;
    }
}
element delete(ArrayListType *L, int pos){
    element item;
```

```
    if(pos < 0 || pos >= L->size)
        error("위치 오류");
    item = L->array[pos];
    for(int i = pos; i<(L->size -1); i++)
        L->array[i] = L->array[i+1];
    L->size--;
    return item;
}
void print_list(ArrayListType *L){
    for(int i =0; i < L->size; i++)
        printf("%d-> ", L->array[i]);
    printf("\n");
}

int main(){
    ArrayListType list;

    init(&list);
    insert(&list, 0, 10); print_list(&list);
    insert(&list, 0, 20); print_list(&list);
    insert(&list, 0, 30); print_list(&list);
    insert_last(&list, 40); print_list(&list);
    delete(&list, 0); print_list(&list);
    return 0;
}
```

▼ 연결 리스트

▼ 배열 형식의 리스트 구현 단점

- 정적 변수로 선언 → 크기를 정하기 힘듦
  - 매우 큰 크기로 선언 → 메모리 낭비가 심함
- 삽입/삭제 비용이 큼

- 항상 연속적인 메모리 공간으로 구현되기 때문에 리스트 끝에서 삽입/삭제가 일어나지 않을 경우, 리스트 데이터의 많은 부분을 밀거나 당기는 연산이 필요함

▼ 연결된 표현

- 장점 : 동적으로 크기가 변할 수 있고, 삭제/삽입 시 데이터들을 이동할 필요가 없음
- 리스트의 항목들을 노드라고 하는 곳에 분산하여 저장
- 노드는 데이터 필드와 링크 필드로 구성
  - data field : 리스트의 원소, 즉 데이터의 값을 저장하는 곳
  - link field : 다른 노드의 주소 값을 저장하는 장소 → **포인터**
- 장점
  - 삽입, 삭제가 보다 용이함 → 연산 오버헤드가 적다
  - 연속된 메모리 공간이 필요 없다
  - 크기 제한이 없다
- 단점
  - 구현이 어렵다
  - 오류가 발생하기 쉽다

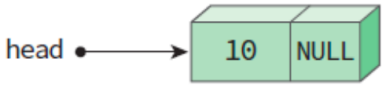
▼ 단순 연결 리스트

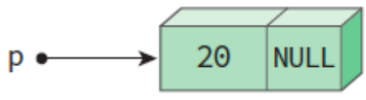
- 하나의 링크 필드를 이용하여 연결
- 마지막 노드의 링크 값은 NULL

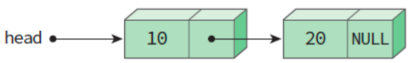
```
typedef int element;
typedef struct ListNode{
    element data;
    struct ListNode *link;
}ListNode
```


```
//두 번째 노드 생성
ListNode *p;
p = (ListNode *)malloc(sizeof(ListNode));
p→data = 20;
p→link = NULL

//node link
head→link = p;
```







▼ 단순 연결 리스트의 연산

- insert_first() : 리스트의 시작 부분에 항목을 삽입함
- insert() : 리스트의 중간 부분에 항목을 삽입함
- delete_first() : 리스트의 첫번째 항목을 삭제함
- delete() : 리스트의 중간 항목을 삭제함
- print_list() : 리스트를 방문하여 모든 항목을 출력함

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

#define MAX 100

typedef int element;
typedef struct ListNode{
    element data;
    struct ListNode *link;
}ListNode;
typedef struct{
    ListNode *head;
    ListNode *tail;
    int size;
};

ListNode *insert_first(ListNode *head, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = head;
    head = p;
    return head;
}
ListNode *insert(ListNode *head, ListNode *pre, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = pre→link;
    pre→link = p;
    return head;
}
ListNode *delete_first(ListNode *head){
```

```
    ListNode *removed;
    if(head == NULL) return NULL;
    removed = head;
    head = removed→link;
    free(removed);
    return head;
}
ListNode *delete(ListNode *head, ListNode *pre){
    ListNode *removed;
    if(removed == NULL) return head;
    removed = pre→link;
    pre→link = removed→link;
    free(removed);
    return head;
}
void print_list(ListNode *head){
    ListNode *p;
    for(p = head; p!= NULL; p = p→link)
        printf("%d→ ", p→data);
    printf("NULL\n");
}
int main(){
    ListNode *head = NULL;
    for(int i = 0; i < 5; i++){
        head = insert_first(head, i);
        print_list(head);
    }
    for(int i = 0; i < 5; i++){
        head = delete_first(head);
        print_list(head);
    }
}
```

▼ 단어 저장 연결리스트

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100

typedef struct{
```

```
    char name[100];
}element;

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

ListNode *insert_first(ListNode *head, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = head;
    head = p;
    return head;
}
ListNode *insert(ListNode *head, ListNode *pre, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = pre→link;
    pre→link = p;
    return head;
}

void print_list(ListNode *head){
    for(ListNode *p = head; p != NULL; p = p→link)
        printf("%s→ ", p→data.name);
    printf("NULL\n");
}

int main(){
    ListNode *head = NULL;
    element data;

    strcpy(data.name, "APPLE");
    head = insert_first(head, data);
    print_list(head);

    strcpy(data.name, "KIWI");
    head = insert_first(head, data);
    print_list(head);
```

```
    strcpy(data.name, "BANANA");
    head = insert_first(head, data);
    print_list(head);

    return 0;
}
```

▼ 특정 값 탐색

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

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

ListNode *insert_first(ListNode *head, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = head;
    head = p;
    return head;
}
ListNode *print_list(ListNode *head){
    ListNode *p;
    for(p = head; p!=NULL; p = p→link)
        printf("%d→ ", p→data);
    printf("NULL\n");
}
ListNode *search_list(ListNode *head, element x){
    ListNode *p;

    while(p != NULL){
        if(p→data == x) return p;
        p = p→link;
    }
    return NULL;
}
```

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

    head = insert_first(head, 10);
    print_list(head);
    head = insert_first(head, 20);
    print_list(head);
    head = insert_first(head, 30);
    print_list(head);
    if(search_list(head, 30) != NULL)
        printf("30을 찾았습니다.\n");
    else
        printf("30을 찾지 못했습니다.\n");

    return 0;
}
```

▼ 2개의 리스트를 연결

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

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

ListNode *insert_first(ListNode *head, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p->data = value;
    p->link = head;
    head = p;
    return head;
}
void print_list(ListNode *head){
    ListNode *p;
    for(p = head; p!=NULL; p = p->link)
        printf("%d-> ", p->data);
```

```
    printf("NULL\n");
}

ListNode *concat_list(ListNode *head1, ListNode *head2){
    if(head1 == NULL) return head2;
    else if(head2 == NULL) return head2;
    else{
        ListNode *p;
        p = head1;
        while(p→link != NULL)
            p = p→link;
        p→link = head2;
        return head1;
    }
}

int main(){
    ListNode *head1 = NULL;
    ListNode *head2 = NULL;

    head1 = insert_first(head1, 10);
    head1 = insert_first(head1, 20);
    head1 = insert_first(head1, 30);
    print_list(head1);

    head2 = insert_first(head2, 40);
    head2 = insert_first(head2, 50);
    print_list(head2);

    ListNode *total = concat_list(head1, head2);
    print_list(total);
    return 0;
}
```

▼ reverse list

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

typedef int element;
typedef struct ListNode{
```

```
    element data;
    struct ListNode *link;
}ListNode;

ListNode *insert_first(ListNode *head, element value){
    ListNode *p = (ListNode *)malloc(sizeof(ListNode));
    p→data = value;
    p→link = head;
    head = p;
    return head;
}
void *print_list(ListNode *head){
    ListNode *p;
    for(p = head; p != NULL; p = p→link)
        printf("%d→ ", p→data);
    printf("NULL\n");
}
ListNode *reverse(ListNode *head){
    ListNode *p, *q, *r;

    p = head;
    q = NULL;
    while(p != NULL){
        r = q;
        q = p;
        p = p→link;
        q→link = r;
    }
    return q;
}

int main(){
    ListNode *head1 = NULL;
    ListNode *head2 = NULL;

    head1 = insert_first(head1, 10);
    head1 = insert_first(head1, 20);
    head1 = insert_first(head1, 30);
    print_list(head1);

    head2 = reverse(head1);
```

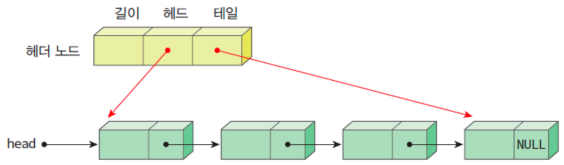
```
    print_list(head2);

    return 0;
}
```

▼ 연결리스트의 응용 - 다항식

- 다항식을 컴퓨터로 처리하기 위한 자료 구조
- 하나의 다항식을 하나의 연결리스트로 표현

```
typedef struct ListNode{
    int coef;
    int expon;
    struct ListNode *link;
} ListNode;
```



▼ polynomial.c

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

typedef struct ListNode{
    int coef;
    int expon;
    struct ListNode *link;
}ListNode;

typedef struct ListType{
    int size;
    ListNode *head;
    ListNode *tail;
}ListType;

void error(){
    fprintf(stderr, "메모리 할당 오류\n");
    exit(1);
}

ListType *create(){
    ListType *plist = (ListType *)malloc(sizeof(ListType));
```

```
    plist->size = 0;
    plist->head = NULL;
    plist->tail = NULL;
    return plist;
}

void insert_last(ListType *plist, int coef, int expon){
    ListNode *temp = (ListNode *)malloc(sizeof(ListNode));
    if(temp == NULL) error("메모리 할당 오류");
    temp->coef = coef;
    temp->expon = expon;
    temp->link = NULL;
    if(plist->tail == NULL)
        plist->head = plist->tail = temp;
    else{
        plist->tail->link = temp;
        plist->tail = temp;
    }
    plist->size++;
}

void poly_print(ListType *plist){
    ListNode *p = plist->head;
    printf("polynomial = ");
    for(;p;p = p->link)
        printf("%dx^%d + ", p->coef, p->expon);
    printf("\n");
}

void poly_add(ListType *plist1, ListType *plist2, ListType *plist3){
    ListNode *a = plist1->head;
    ListNode *b = plist2->head;
    int sum;

    while(a && b){
        if(a->expon == b->expon){
            sum = a->coef + b->coef;
            if(sum != 0) insert_last(plist3, sum, a->expon);
            a = a->link; b = b->link;
        }else if(a->expon > b->expon){
            insert_last(plist3, a->coef, a->expon);
```

```
            a = a->link;
        }else{
            insert_last(plist3, b->coef, b->expon);
            b = b->link;
        }
    }

    for(; a != NULL; a = a->link)
        insert_last(plist3, a->coef, a->expon);
    for(; b != NULL; b = b->link)
        insert_last(plist3, b->coef, b->expon);
}

int main(){
    ListType *list1, *list2, *list3;

    list1 = create();
    list2 = create();
    list3 = create();

    insert_last(list1, 3, 12);
    insert_last(list1, 2, 8);
    insert_last(list1, 1, 0);

    insert_last(list2, 8, 12);
    insert_last(list2, -3, 10);
    insert_last(list2, 10, 6);

    poly_print(list1);
    poly_print(list2);

    poly_add(list1, list2, list3);
    poly_print(list3);

    free(list1);
    free(list2);
    free(list3);
}
```