데이터 구조 5장 실습과제

20223100 박신조

6.1 배열리스트

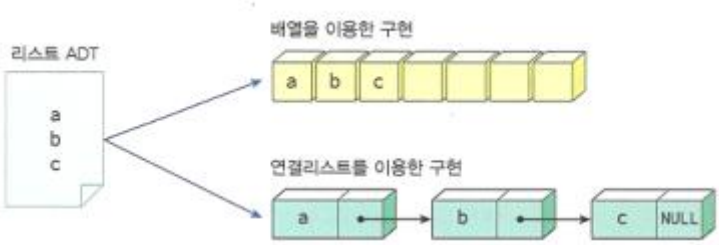
```c
#include <stdio.h>
#include <stdlib.h>
#define max_list_size 100

typedef int element;
typedef struct {
    element array[max_list_size];
    int size;
}arraylisttype;

void error(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_list_size; }

element get_entry(arraylisttype* l, int pos){
    if (pos < 0 || pos >= l->size)
        error("위치 오류");
    return l->array[pos];
}

void print_list(arraylisttype* l){
    int i;
    for (i = 0; i < l->size; i++)
        printf("%d -> ", l->array[i]);
    printf("₩n");
}

void insert_last(arraylisttype* l, element item){
    if (l->size >= max_list_size)
        error("스택 오버 플로우");
    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;
}

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;
}
```

```
//6.1 배열리스트 결과값
10 ->
20 -> 10 ->
30 -> 20 -> 10 ->
30 -> 20 -> 10 -> 40 ->
20 -> 10 -> 40 ->
```

배열을 사용하여 선형 리스트를 표현한 코드입니다.
리스트란 노드들로 이루어진 집합으로 노드에는 데이터, 주소가 저장됩니다.
위 코드는 데이터 삽입, 삭제, 출력이 가능 합니다.

#define max_list_size 100

max_list_size를 100으로 지정



typedef int element

element의 타입을 int로 지정

typedef struct { . . . }arraylisttype

구조체 arraylisttype 선언

멤버 - 데이터를 저장할 arr[], 현재 저장된 데이터의 수 size

void error(char* message)

에러 메시지 message를 출력하고 프로그램 종료

ex) 범위 초과, 리스트가 꽉 찼을 때

void init(arraylisttype* l)

리스트의 초기 설정을 해주는 함수

리스트 원소의 수 초기화

int is_empty(arraylisttype* l)

리스트가 비었는지 확인하는 함수

공백이라면 1, 아니면 거짓 0

int is_full(arraylisttype* l)

리스트가 가득 차있는지 확인하는 함수

size가 max_list_size라면 1, 아니면 거짓 0

element get_entry(arraylisttype* l, int pos)

주어진 위치 pos의 원소를 반환하는 함수

유효하지 않은 인덱스면 에러

ex) 배열의 크기보다 오버, 음수의 값

void print_list(arraylisttype* l)

리스트에 저장된 모든 원소를 순서대로 출력하는 함수

void insert_last(arraylisttype* l, element item)

리스트의 맨 끝에 값을 추가하는 함수

인덱스 size 위치에 item값 저장 후 size + 1

void insert(arraylisttype* l, int pos, element item)

리스트의 pos 위치에 item값 저장하는 함수

리스트의 마지막 인덱스값을 마지막 인덱스 + 1위치로 이동하여

pos위치의 값을 pos+1 위치로 이동 후 pos위치에 item값 추가

element delete(arraylisttype* l, int pos)

리스트의 pos 위치에 저장된 데이터 삭제

이후 pos+1에 저장된 값을 pos 위치로 이동하여 size-1 위치의 값을 size-2 로 이동해준 후 size-1

int main()

insert함수, insert_last함수, delete함수를 호출하여 맨 압에 값 추가, 맨 뒤에 값 추가, 맨 앞 데이터 삭제

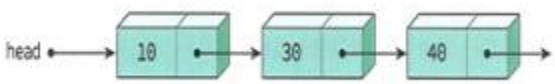
리스트를 변경 할때마다 print_list함수를 호출하여 리스트의 현재 상태를 출력함

6.2 연결리스트

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

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

void error(char* message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}
ListNode* insert_first(ListNode* head, int 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;
    removed = pre->link;
    pre->link = removed->link;
    free(removed);
    return head;
}

void print_list(ListNode* head){
    for (ListNode* p = head; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL \n");
}

int main(void){
    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);
    }
    return 0;
}
```

```
//6.2 연결리스트 결과값
0->NULL
1->0->NULL
2->1->0->NULL
3->2->1->0->NULL
4->3->2->1->0->NULL
3->2->1->0->NULL
2->1->0->NULL
1->0->NULL
0->NULL
NULL
```

구조체를 사용하여 단일 연결 리스트(Singly Linked List) 를 구현한 코드입니다.
구조체는 리스트의 노드 역활을 합니다.



typedef int element
element의 타입을 int로 지정

typedef struct ListNode { . . . } ListNode;
구조체 ListNode 선언 - 노드부분 구현
멤버 - data는 데이터를 저장할 변수, link는 다음 노드를 가리키는 포인터

void error(char* message)

에러 메시지 message를 출력하고 프로그램 종료

ex) 범위 초과, 리스트가 꽉 찼을 때

ListNode* insert_first(ListNode* head, int value)

리스트의 맨 앞에 새로운 노드를 추가하는 함수

새로운 노드를 동적으로 생성하고 value 값 저장, 기존의 head를 새로운 노드의 link로 연결

ListNode* insert(ListNode* head, ListNode* pre, element value)

리스트의 중간에 노드를 추가하는 함수.

새로운 노드를 동적으로 생성하고 value 값 저장, 새 노드의 link를 per->link로 지정하여 다음 노드를 가리키도록 하고

pre->link를 새로 생성된 노드를 가리키도록 설정

ListNode* delete_first(ListNode* head)

리스트의 맨 앞 노드를 삭제하는 함수

head가 가리키는 노드를 삭제하고 그다음 노드를 head로 설정

ListNode* delete(ListNode* head, ListNode* pre)

리스트의 중간 노드를 삭제하는 함수

pre->link가 가리키는 노드를 removed로 지정, pre->link를 removed->link로 변경하여 removed 노드를 리스트에서 삭제

이후 삭제된 노드의 메모리 반환

void print_list(ListNode* head)

리스트 현재 상태를 출력하는 함수

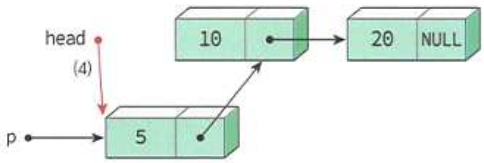
임시 구조체 p를 선언하여 head 노드로 지정해 준 후 data 값을 출력, p를 p->link로 지정 이후 p가 null 일때 종료.

int main(void)

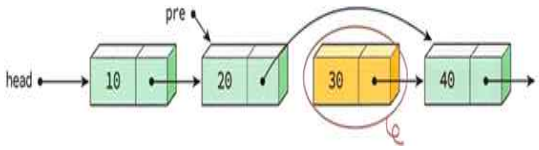
리스트가 공백 상태이기 때문에 head를 null 값으로 초기화

insert_first 함수와 delete_first 함수를 사용하여 리스트에 값을 추가하고 삭제하며

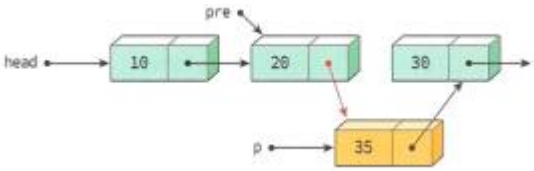
매번 리스트가 변동될 때마다 print_list 함수를 통해 리스트를 출력



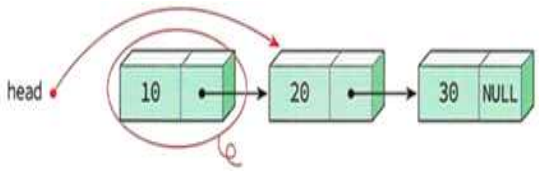
insert_first 함수



delete_first 함수



insert 함수



delete 함수

Lab1 단어들을 저장하는 연결리스트

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

typedef struct {
    char name[100];
}element;

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

void error(char* message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}

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

int main(){
    printf("//Lab1 단어들을 저장하는 연결리스트 결과값\n");
    ListNode* head = NULL;
    element data;

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

    strcpy_s(data.name, sizeof(data.name), "KIWI");
    head = insert_first(head, data);
    print_list(head);

    strcpy_s(data.name, sizeof(data.name), "BANANA");
    head = insert_first(head, data);
    print_list(head);
    return 0;
}
```

```
//Lab1 단어들을 저장하는 연결리스트 결과값
APPLE->NULL
KIWI->APPLE->NULL
BANANA->KIWI->APPLE->NULL
```

6-2에서 구현한 리스트의 데이터 타입을 int 형에서 char 형으로 변경하여 문자열을 저장하는 단일 연결 리스트를 구현한 코드입니다.

typedef struct { . . . } element

문자열을 저장할 변수를 구조체 element로 설정
맴버 - 크기가 100인 char형 배열

typedef struct ListNode { . . . } ListNode

구조체 ListNode 선언 - 노드 부분 구현
멤버 - data는 문자열 데이터를 저장할 변수, link는 다음 노드를 가리키는 포인터

void error(char* message)

에러 메시지 message를 출력하고 프로그램 종료
ex) 범위 초과, 리스트가 꽉 찼을 때

ListNode* insert_first(ListNode* head, element value)

리스트의 맨 앞에 새로운 노드를 추가하는 함수
새로운 노드를 동적으로 생성하고 value 값 저장, 기존의 head를 새로운 노드의 link로 연결
이때 value를 data에 저장할 때 strcpy_s(문자열을 복사 해주는 함수)를 사용하는 것이 정석에 가깝지만
p->data = value를 사용해도 컴파일 에러가 발생하지 않습니다. 이유는 C언어에서 구조체는 통째로 복사 가능합니다.
strcpy_s는 깊은 복사, p->data = value 얕은 복사에 해당

void print_list(ListNode* head)

리스트 현재 상태를 출력하는 함수

임시 구조체 p를 선언하여 head 노드로 지정해 준 후 data 값을 출력, p를 p->link로 지정 이후 p가 null 일때 종료.

int main(void)

리스트가 공백 상태이기 때문에 head를 null 값으로 초기화

리스트에 추가할 문자열을 strcpy_s를 통해 구조체 data에 저장

insert_first 함수를 사용하여 리스트에 값을 추가

매번 리스트가 변동될 때마다 print_list 함수를 통해 리스트를 출력

Lab2 특정한 값을 탐색하는 함수

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

typedef int element;
typedef struct {
    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){
    for (ListNode* p = head; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL \n");
}

ListNode* search_list(ListNode* head, element x){

    ListNode* p = head;
    while (p != NULL) {
        if (p->data == x) return p;
        p = p->link;
    }
    return NULL;
}

int main(){
    printf("Lab2 특정한 값을 탐색하는 함수\n");
    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("리스트에서 30을 찾았습니다.\n");
    else
        printf("리스트에서 30을 찾지 못했습니다.\n");
    return 0;
}
```

```
Lab2 특정한 값을 탐색하는 함수 결과값
10->NULL
20->10->NULL
30->20->10->NULL
리스트에서 30을 찾았습니다.
```

typedef int element

element 타입을 int로 지정합니다.

typedef struct ListNode { . . . } ListNode

구조체 ListNode 선언 - 노드부분 구현

멤버 - data는 데이터를 저장할 변수, link는 다음 노드를 가리키는 포인터

ListNode* insert_first(ListNode* head, element value)

리스트의 맨 앞에 새로운 노드를 추가하는 함수

새로운 노드를 동적으로 생성하고 value값 저장, 기존의 head를 새로운 노드의 link로 연결

void print_list(ListNode* head)

리스트 현재 상태를 출력하는 함수

임시 구조체 p를 선언하여 head 노드로 지정 해준 후 data 값을 출력, p를 p->link로 지정 이후 p가 null일때 종료.

ListNode* search_list(ListNode* head, element x)

리스트에 있는 특정한 값을 찾는 함수

리스트의 노드를 순회하며 x 값을 가진 노드를 찾고 구조체 p를 리턴

리스트에 찾는 요소가 없다면 NULL 반환

print_list 함수와 작동 방식이 비슷함

int main(void)

리스트가 공백 상태이기 때문에 head를 null 값으로 초기화

insert_first 함수를 사용하여 리스트에 값을 추가

매번 리스트가 변동될 때마다 print_list 함수를 통해 리스트를 출력합니다.

이후 search_list 함수를 호출해서 찾으려는 값이 있는지 판단 후 결과 값 출력

Lab3 두 개의 리스트를 하나로 합치는 함수

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

typedef int element;
typedef struct {
    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){
    for (ListNode* 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 head1;
    else {
        ListNode* p;
        p = head1;
        while (p->link != NULL)
            p = p->link;
        p->link = head2;
        return head1;
    }
}

int main(){
    printf("Lab3 두 개의 리스트를 하나로 합치는 함수 결과값\n");
    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;
}
```

```
Lab3 두 개의 리스트를 하나로 합치는 함수 결과값
30->20->10->NULL
50->40->NULL
30->20->10->50->40->NULL
```

typedef int element

element 타입을 int로 지정합니다.

typedef struct ListNode { . . . } ListNode

구조체 ListNode 선언 - 노드부분 구현

멤버 - data는 데이터를 저장할 변수, link는 다음 노드를 가리키는 포인터

ListNode* insert_first(ListNode* head, element value)

리스트의 맨 앞에 새로운 노드를 추가하는 함수

새로운 노드를 동적으로 생성하고 value값 저장, 기존의 head를 새로운 노드의 link로 연결

void print_list(ListNode* head)

리스트 현재 상태를 출력하는 함수

임시 구조체 p를 선언하여 head 노드로 지정 해준 후 data 값을 출력, p를 p->link로 지정 이후 p가 null일때 종료.

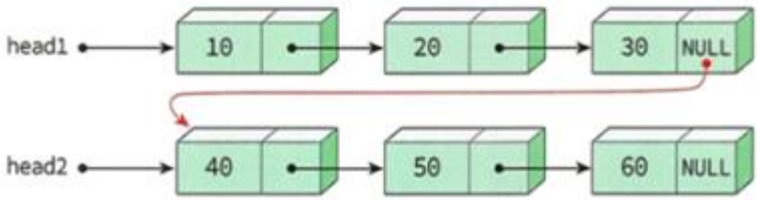
ListNode* concat_list(ListNode *head1, ListNode *head2)

첫 번째 리스트의 끝나는 주소와 두 번째 리스트의 head를 하나로 연결시키는 함수
리스트 1이 NULL이면 리스트 2를, 리스트 2가 NULL이면 리스트 1을 반환.
두 개의 리스트가 모두 공백 상태가 아니라면
두 개의 리스트를 병합시켜 줄 임시 노드를 첫 번째 리스트의 head 노드로 초기화
while 문을 통해 첫 번째 함수의 마지막 노드(null을 가리키는 노드)를 찾은 뒤
두 번째 head 노드를 가리키도록 변경한 후 임시 노드를 리턴

둘 다 있으면 연결 후 리스트1 반환.

int main(void)

두 개의 리스트 head1, head2 선언
리스트가 공백 상태이기 때문에 head1,head2를 null 값으로 초기화
insert_first 함수를 사용하여 리스트에 값을 추가
매번 리스트가 변동될 때마다 print_list 함수를 통해 리스트를 출력합니다.
이후 concat_list 함수를 호출해서 두 개의 리스트를 병합 후 결과 출력



Lab4 리스트를 역순으로 만드는 연산

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

typedef int element;

typedef struct {
    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) {
    for (ListNode* 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() {
    printf("Lab4 리스트를 역순으로 만드는 연산 결과값\n");
    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;
}
```

```
Lab4 리스트를 역순으로 만드는 연산 결과값
30->20->10->NULL
10->20->30->NULL
```

typedef int element

element 타입을 int로 지정합니다.

typedef struct ListNode { . . . } ListNode

구조체 ListNode 선언 - 노드부분 구현

멤버 - data는 데이터를 저장할 변수, link는 다음 노드를 가리키는 포인터

ListNode* insert_first(ListNode* head, element value)

리스트의 맨 앞에 새로운 노드를 추가하는 함수

새로운 노드를 동적으로 생성하고 value 값 저장, 기존의 head를 새로운 노드의 link로 연결

void print_list(ListNode* head)

리스트 현재 상태를 출력하는 함수

임시 구조체 p를 선언하여 head 노드로 지정해 준 후 data 값을 출력, p를 p->link로 지정 이후 p가 null 일때 종료.

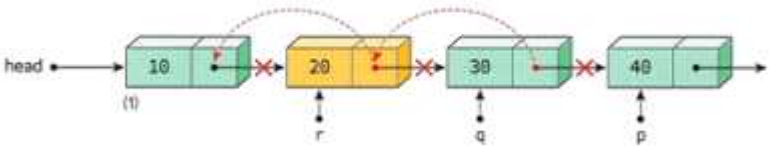
ListNode* reverse(ListNode* head)

노드의 link가 다음 노드를 가리키던 주소를 이전 노드를 가리키도록 변경시키는 함수(역순으로 변경)

p: 현재 노드
q: 이전 노드(head)
r: 임시 노드

순서	p	q	r	
1	30	NULL	NULL	q->link = r → 30->NULL
2	20	30	NULL	q->link = r → 20->30
3	10	20	30	q->link = r → 10->20

결과
30->20->10->null
reverse(head)
10->20->30->null



int main(void)

리스트를 역순으로 변경한 후 비교하기 위해 리스트 head1, head2 두 개 선언
리스트가 공백 상태이기 때문에 head1, head2를 null 값으로 초기화
insert_first 함수를 사용하여 리스트에 값을 추가
매번 리스트가 변동될 때마다 print_list 함수를 통해 리스트를 출력합니다.
이후 reverse 함수를 호출해서 기존의 리스트를 역순으로 변경 후 출력

6.9 연결리스트로 구현한 다항식 덧셈

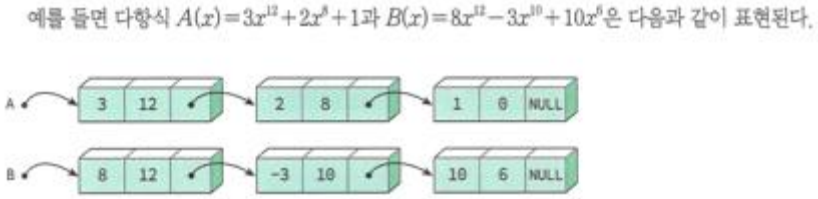
```c
#include <stdio.h>
#include <stdlib.h>

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

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

void error(char* message) {
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

ListType* create( ) {
    ListType* plist = (ListType*)malloc(sizeof(ListType));
    plist->size = 0;
    plist->head = 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 에러");
    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_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);
}

void poly_print(ListType* plist) {
    ListNode* p = plist->head;
    printf("polynomial = ");
    while (p != NULL) {
        printf("%d^%d + ", p->coef, p->expon);
        p = p->link;
    }
    printf("₩n");
}
```

```c
int main() {
    printf("6.9 연결리스트로 구현한 다항식 덧셈 결과값₩n");
    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);
    return 0;
```

```
6.9 연결리스트로 구현한 다항식 덧셈 결과값
polynomial = 3^12 + 2^8 + 1^0 +
polynomial = 8^12 + -3^10 + 10^6 +
polynomial = 11^12 + -3^10 + 2^8 + 10^6 + 1^0 +
```

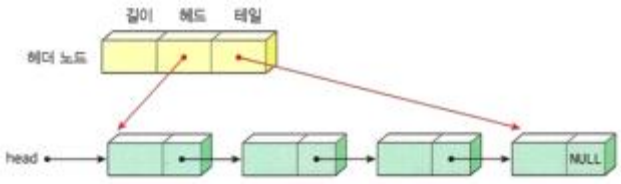
연결 리스트를 이용해 다항식을 표현하고, 두 다항식을 더한 결과를 출력하는 코드입니다.
구조체를 이용해 각 항(term)을 연결 리스트로 구성하고
항들의 지수를 기준으로 정렬된 상태에서 덧셈 연산을 수행할 수 있습니다.

typedef struct ListNode { . . . } ListNode;
하나의 항(term)을 구현한 구조체
멤버 - coef는 계수, expon은 지수, link는 다음 항



typedef struct { . . . } ListType;
다항식을 구현한 구조체
멤버 - head는 첫 항, tail은 마지막 항, size는 노드 개수

ListType* create()
다항식 리스트를 생성하고 초기화해 주는 함수



void insert_last(ListType* plist, int coef, int expon)
새로운 항을 동적으로 할당하여 리스트에 추가해 주는 함수
리스트가 비어있다면 head와 tail 모두 설정,
그렇지 않다면 tail->link를 갱신.

void poly_add(ListType* plist1, ListType* plist2, ListType* plist3)
두 다항식을 연산해주는 함수
지수를 기준으로 두 다항식을 비교하며 덧셈연산 진행
지수가 같으면 계수끼리 더하여 저장
두 항의 지수가 다르다면 더 큰 항을 저장
이후 두 다항식 중 하나라도 null값(마지막 항)이 되면 while문 종료
만약 두 다항식이 남아있다면 남은 식을 for문을 통해 저장

void poly_print(ListType* plist)
다항식을 출력하는 함수
예: 3^12 + 2^8 + 1^0 +

int main(void)
다항식 리스트 list1, list2, list3 을 create 함수를 호출하여 초기화해 주고 list1, list2 다항식을 연산한 결과를 list3에 저장하여 출력합니다.
insert_last함수를 통해 다항식을 리스트에 추가합니다. 이후 두 다항식을 poly_print 함수를 통해 다항식을 출력합니다.
poly_add함수를 통해 두 다항식 덧셈 연산을 하여 list3에 값을 저장한 후 출력합니다. 모든 작업이 끝난 후 할당받은 메모리를 반납합니다.