/\*

2404010 연결리스트

\*/

1. **리스트를 하나로 결합하는 함수 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;

ListNode\* insert\_first(ListNode\* head, element value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;// (2)

p->link = head;// 헤드 포인터의 값을 복사//(3)

head = p;// 헤드 포인터 변경//(4)

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 head2;

else {

ListNode\* p;

p = head1;

while (p->link != NULL)

p = p->link;

p->link = head2;

return head1;

}

}

// 테스트 프로그램

int main(void)

{

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;

}

2. **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;

ListNode\* insert\_first(ListNode\* head, element value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;// (2)

p->link = head;// 헤드 포인터의 값을 복사//(3)

head = p;// 헤드 포인터 변경//(4)

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)

{

// 순회 포인터로 p, q, r을 사용

ListNode\* p, \* q, \* r;

p = head; // p는 역순으로 만들 리스트

q = NULL; // q는 역순으로 만들 노드

while (p != NULL) {

r = q; // r은 역순으로 된 리스트.

// r은 q, q는 p를 차례로 따라간다.

q = p;

p = p->link;

q->link = r; // q의 링크 방향을 바꾼다.

}

return q;

}

// 테스트 프로그램

int main(void)

{

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;

}

3. polynomical

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

typedef struct ListNode { // 노드 타입

int coef;

int expon;

struct ListNode\* link;

} ListNode;

// 연결 리스트 헤더

typedef struct ListType { // 리스트 헤더 타입

int size;

ListNode\* head;

ListNode\* tail;

} ListType;

// 오류 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

// 리스트 헤더 생성 함수

ListType\* create()

{

ListType\* plist = (ListType\*)malloc(sizeof(ListType));

plist->size = 0;

plist->head = plist->tail = NULL;

return plist;

}

// plist는 연결 리스트의 헤더를 가리키는 포인터, coef는 계수,

// expon는 지수

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++;

}

// list3 = list1 + list2

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) { // a의 차수 > b의 차수

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) { // a의 차수 == b의 차수

insert\_last(plist3, a->coef, a->expon);

a = a->link;

}

else { // a의 차수 < b의 차수

insert\_last(plist3, b->coef, b->expon);

b = b->link;

}

}

// a나 b중의 하나가 먼저 끝나게 되면 남아있는 항들을 모두

// 결과 다항식으로 복사

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 = ");

for (; p; p = p->link) {

printf("%d^%d + ", p->coef, p->expon);

}

printf("\n");

}

//

int main(void)

{

ListType\* list1, \* list2, \* list3;

// 연결 리스트 헤더 생성

list1 = create();

list2 = create();

list3 = create();

// 다항식 1을 생성

insert\_last(list1, 3, 12);

insert\_last(list1, 2, 8);

insert\_last(list1, 1, 0);

// 다항식 2를 생성

insert\_last(list2, 8, 12);

insert\_last(list2, -3, 10);

insert\_last(list2, 10, 6);

poly\_print(list1);

poly\_print(list2);

// 다항식 3 = 다항식 1 + 다항식 2

poly\_add(list1, list2, list3);

poly\_print(list3);

free(list1); free(list2); free(list3);

}