자료구조

- 과제번호1 –

학번: 20202878

이름: 조성주

**1. 하노이의 탑**

**1-A. 소스 코드**

#include <stdio.h>

void HanoiTowerMove(int num, char from, char by, char to) {

if (num == 1) { //이동할 원판의 수 1개인 경우

printf("원판1을 %c에서 %c로 이동\n", from, to);

}

else {

HanoiTowerMove(num - 1, from, to, by); //재귀함수, to와 by 원판 바꿈

printf("원판%d을 %c에서 %c로 이동\n", num, from, to);

HanoiTowerMove(num - 1, by, from, to); //재귀함수, from과 by 원판 바꿈

}

}

int main() {

HanoiTowerMove(4, 'A', 'B', 'C'); //4개의 원판을 'A'에서 'C'로 이동

return 0;

}

**1-B. 소스 코드 실행 방법**

컴파일 후 실행

**1-C. 출력 결과**



**2. 희소행렬 전치행렬로 나타내기**

**2-A. 소스 코드**

#include <stdio.h>

typedef struct{

int row;

int col;

int value;

} term;

void smTranspose(term a[], term b[]) {

int m, n, v, i, j, p;

m = a[0].row; //희소행렬 a의 행 수

n = a[0].col; //희소행렬 a의 열 수

v = a[0].value; //희소행렬 a에서 0이 아닌 원소 수

b[0].row = n; //전치행렬 b의 행 수

b[0].col = m; //전치행렬 b의 열 수

b[0].value = v; //전치행렬 b의 원소 수

if (v > 0) { //0이 아닌 원소가 있는 경우에만 전치 연산 수행

p = 1;

for (i = 0; i < n; i++) {

//희소행렬 a의 열별로 전치 반복 수행

for (j = 1; j <= v; j++) {

//0이 아닌 원소 수에 대해서만 반복 수행

if (a[j].col == i) {

//현재의 열에 속하는 원소가 있으면 b[]에 삽입

b[p].row = a[j].col;

b[p].col = a[j].row;

b[p].value = a[j].value;

p++;

}

}

}

}

}

int main() {

//파일 오픈

FILE \*ifp;

FILE \*ofp;

ifp = fopen("input.txt", "r");

ofp = fopen("output.txt", "w");

int i, j;

term a[9];

term b[9];

//fscanf를 통해 input.txt에서 값 받아오기

for (i = 0; i < 9; i++) {

fscanf(ifp, "%d", &a[i].row);

fscanf(ifp, "%d", &a[i].col);

fscanf(ifp, "%d", &a[i].value);

}

smTranspose(a, b);

//fprintf를 통해 output.txt에 출력하기

for (i = 0; i < 9; i++)

fprintf(ofp, "%d %d %d\n", b[i].row, b[i].col, b[i].value);

fclose(ifp);

fclose(ofp);

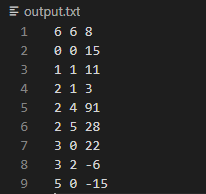
return 0;

}

**2-B. 소스 코드 실행 방법**

컴파일 후 실행

**2-C. 출력 결과**



**3. 순차 리스트의 다항식 표현방법 2를 이용, 다항식 계산**

**3-A. 소스 코드**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_TERMS 101

typedef struct {

float coef;

int expon;

} polynomial;

polynomial terms[MAX\_TERMS] = {{3, 14}, {2, 8}, {1, 0}, {8, 14}, {-3, 10}, {10, 6}};

int avail = 6; //현재 비어있는 요소의 인덱스

//두 개의 정수를 비교

char compare(int a, int b) {

if (a > b) return '>';

else if (a == b) return '=';

else return '<';

}

//새로운 항을 다항식에 추가

void attach(float coef, int expon) {

if (avail > MAX\_TERMS) {

fprintf(stderr, "항의 개수가 너무 많음\n");

exit(1);

}

terms[avail].coef = coef;

terms[avail].expon = expon;

avail++;

}

void poly\_add2(int As, int Ae, int Bs, int Be, int \*Cs, int \*Ce) {

float tempcoef;

\*Cs = avail;

while (As <= Ae && Bs <= Be)

switch (compare(terms[As].expon, terms[Bs].expon)) {

case '>': //A의 차수 > B의 차수 -> A만 이동

attach(terms[As].coef, terms[As].expon);

As++;

break;

case '=': //A의 차수 == B의 차수 -> 덧셈 진행

tempcoef = terms[As].coef + terms[Bs].coef;

if (tempcoef)

attach(tempcoef, terms[As].expon);

As++; Bs++;

break;

case '<': //A의 차수 < B의 차수 -> B만 이동

attach(terms[Bs].coef, terms[Bs].expon);

Bs++;

break;

}

//A와 B의 나머지 항들 이동

for(; As <= Ae; As++)

attach(terms[As].coef, terms[As].expon);

for (; Bs <= Be; Bs++)

attach(terms[Bs].coef, terms[Bs].expon);

\*Ce = avail - 1;

}

void print\_poly(int s, int e) {

for (int i = s; i < e; i++)

printf("%3.1fx^%d +", terms[i].coef, terms[i].expon);

printf("%3.1fx^%d\n", terms[e].coef, terms[e].expon);

}

int main() {

int As = 0, Ae = 2, Bs = 3, Be = 5, Cs, Ce;

poly\_add2(As, Ae, Bs, Be, &Cs, &Ce);

print\_poly(As, Ae);

print\_poly(Bs, Be);

print\_poly(Cs, Ce);

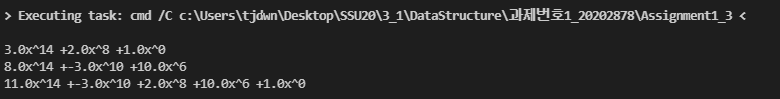
return 0;

}

**3-B. 소스 코드 실행 방법**

컴파일 후 실행

**3-C. 출력 결과**



**4. 단순 연결리스트 이용, 다항식 계산**

**4-A. 소스 코드**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

float coef;

int expon;

struct node\* link;

} listnode;

//노드 연결

listnode\* insert(listnode\* head, float coef, int expon) {

listnode\* temp = (listnode\*) malloc(sizeof(listnode));

temp->coef = coef;

temp->expon = expon;

temp->link = NULL;

if (head == NULL || expon > head->expon) {

temp->link = head;

head = temp;

}

else {

listnode\* temp2;

temp2 = head;

while (temp2->link != NULL && temp2->link->expon > expon)

temp2 = temp2->link;

temp->link = temp2->link;

temp2->link = temp;

}

return head;

}

void poly\_print(listnode\* head) {

listnode\* temp = head;

while(temp != NULL) {

printf("%3.1fx^%d", temp->coef, temp->expon);

temp = temp->link;

if (temp != NULL)

printf(" + ");

else printf("\n");

}

}

void poly\_add(listnode\* head1, listnode\* head2) {

listnode\* a = head1;

listnode\* b = head2;

listnode\* head3 = NULL;

while (a != NULL && b != NULL) {

if (a->expon == b -> expon) { //A의 차수 == B의 차수 -> 덧셈 진행

head3 = insert(head3, a->coef + b->coef, a->expon);

a = a->link;

b = b->link;

}

else if (a -> expon > b -> expon) { //A의 차수 > B의 차수 -> A만 이동

head3 = insert(head3, a->coef, a->expon);

a = a->link;

}

else if (a -> expon < b -> expon) { //A의 차수 < B의 차수 -> B만 이동

head3 = insert(head3, b -> coef, b -> expon);

b = b->link;

}

}

while(a != NULL) {

head3 = insert(head3, a->coef, a->expon);

a = a->link;

}

while (b != NULL) {

head3 = insert(head3, b->coef, b->expon);

b = b->link;

}

poly\_print(head3);

}

int main() {

listnode\* head1 = NULL;

listnode\* head2 = NULL;

listnode\* head3 = NULL;

//head1 연결리스트 생성

head1 = insert(head1, 3, 14);

head1 = insert(head1, 2, 8);

head1 = insert(head1, 1, 0);

//head2 연결리스트 생성

head2 = insert(head2, 8, 14);

head2 = insert(head2, -3, 10);

head2 = insert(head2, 10, 6);

poly\_print(head1);

poly\_print(head2);

poly\_add(head1, head2);

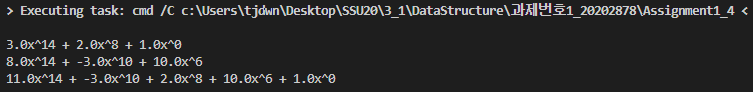
return 0;

}

**4-B. 소스 코드 실행 방법**

컴파일 후 실행

**4-C. 출력 결과**

****

**5. 단순 연결리스트 이용, 다항식 뺄셈 계산**

**5-A. 소스 코드**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

float coef;

int expon;

struct node\* link;

} listnode;

//노드 연결

listnode\* insert(listnode\* head, float coef, int expon) {

listnode\* temp = (listnode\*) malloc(sizeof(listnode));

temp->coef = coef;

temp->expon = expon;

temp->link = NULL;

if (head == NULL || expon > head->expon) {

temp->link = head;

head = temp;

}

else {

listnode\* temp2;

temp2 = head;

while (temp2->link != NULL && temp2->link->expon > expon)

temp2 = temp2->link;

temp->link = temp2->link;

temp2->link = temp;

}

return head;

}

void poly\_print(listnode\* head) {

listnode\* temp = head;

while(temp != NULL) {

printf("%3.1fx^%d", temp->coef, temp->expon);

temp = temp->link;

if (temp != NULL)

printf(" + ");

else printf("\n");

}

}

void poly\_sub(listnode\* head1, listnode\* head2) {

listnode\* a = head1;

listnode\* b = head2;

listnode\* head3 = NULL;

while (a != NULL && b != NULL) {

if (a->expon == b -> expon) { //A의 차수 == B의 차수 -> 뺄셈 진행

head3 = insert(head3, a->coef - b->coef, a->expon);

a = a->link;

b = b->link;

}

else if (a -> expon > b -> expon) { //A의 차수 > B의 차수 -> A만 이동

head3 = insert(head3, a->coef, a->expon);

a = a->link;

}

else if (a -> expon < b -> expon) { //A의 차수 < B의 차수 -> B만 이동

head3 = insert(head3, b -> coef, b -> expon);

b = b->link;

}

}

while(a != NULL) {

head3 = insert(head3, a->coef, a->expon);

a = a->link;

}

while (b != NULL) {

head3 = insert(head3, b->coef, b->expon);

b = b->link;

}

poly\_print(head3);

}

int main() {

listnode\* head1 = NULL;

listnode\* head2 = NULL;

listnode\* head3 = NULL;

//head1 연결리스트 생성

head1 = insert(head1, 3, 14);

head1 = insert(head1, 2, 8);

head1 = insert(head1, 1, 0);

//head2 연결리스트 생성

head2 = insert(head2, 8, 14);

head2 = insert(head2, -3, 10);

head2 = insert(head2, 10, 6);

poly\_print(head1);

poly\_print(head2);

poly\_sub(head1, head2);

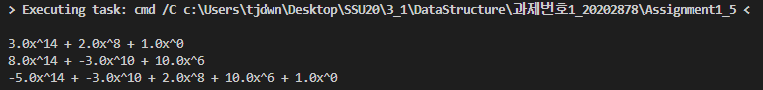
return 0;

}

**5-B. 소스 코드 실행 방법**

컴파일 후 실행

**5-C. 출력 결과**

****