```c
//2020118008 박보경
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.

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

#define MALLOC(p, s) \
        if( !( (p) = (polyPointer)malloc( s ) ) ){      \
                fprintf(stderr, "Insufficient memory");\
                exit(EXIT_FAILURE);\
        }

#define COMPARE(x, y) (((x)<(y)) ? -1 : ((x)==(y)) ? 0 : 1)

// 노드 생성
typedef struct polyNode* polyPointer;
typedef struct polyNode {
        int coef;
        int expon;
        polyPointer link;
}polyNode;
polyPointer a = NULL, b = NULL, c = NULL;

void findLast(polyPointer first, polyPointer* last);
void insert(polyPointer* first, polyPointer x, int coef, int expon);
void printList(polyPointer first);
polyPointer padd(polyPointer a, polyPointer b);
void attach(int coefficient, int exponent, polyPointer* ptr);
void erase(polyPointer* first);
void inputPoly(char* filename, polyPointer* p);

int main(void)
{
        inputPoly("a.txt", &a);
        inputPoly("b.txt", &b);

        //다항식 a를 출력
        printf("a :");
        printList(a);

        printf("b :");
        printList(b);
```

```
	c = padd(a, b);
	printf("a+b=c: ");
	printList(c);

	erase(&a);
	erase(&b);
	erase(&c);

	return 0;
}

void insert(polyPointer* first, polyPointer x, int coef, int expon)
{
	//새로운 노드를 하나 생성
	polyPointer temp;
	MALLOC(temp, sizeof(*temp));
	temp->coef = coef;
	temp->expon = expon;

	//앞에 생성된 노드들이 존재한다면 (first가 비어있지 않으면)
	if (*first)
	{
		if (!x)
		{
			temp->link = *first;
			*first = temp;
		}
		else
		{
			temp->link = x->link;
			x->link = temp;
		}
	}
	else
	{
		temp->link = NULL;
		*first = temp;
	}
}

void inputPoly(char* filename, polyPointer* p)
{
	int coef, expon;
```

```
        polyPointer x = NULL;
        polyPointer last = NULL;
        char order;

        //파일을 읽음
        FILE* fp;
        fopen_s(&fp, filename, "r");
        if (fp == NULL)
        {
                fprintf(stderr, "cannot open the file");
                exit(EXIT_FAILURE);
        }

        //첫줄을 받아옴
        fscanf_s(fp, "%c", &order, sizeof(order));

        if (order == 'd')
        {
                fscanf_s(fp, "%d%d", &coef, &expon);
                while (!feof(fp))
                {
                        findLast(*p, &last);
                        x = last;
                        insert(p, x, coef, expon);
                        fscanf_s(fp, "%d%d", &coef, &expon);
                }
        }
        else
        {
                fscanf_s(fp, "%d%d", &coef, &expon);
                while (!feof(fp))
                {
                        insert(p, NULL, coef, expon);
                        fscanf_s(fp, "%d%d", &coef, &expon);
                }
        }

        fclose(fp);
}

//first가 가리키는 체인의 마지막 노드를 찾는다
void findLast(polyPointer first, polyPointer* last)
{
```

```
        for (; first; first = first->link)
                *last = first;
}

void printList(polyPointer first)
{
        for (; first; first = first->link)
                printf("%+4dx^%d ", first->coef, first->expon);
        printf("\n");
}

polyPointer padd(polyPointer a, polyPointer b)
{
        polyPointer c, rear, temp;
        int sum;
        MALLOC(rear, sizeof(*rear));
        c = rear;

        //a와 b모두 존재한다면
        while (a && b)
        {
                switch (COMPARE(a->expon, b->expon))
                {
                case -1:
                        attach(b->coef, b->expon, &rear);
                        b = b->link;
                        break;
                case 0:
                        sum = a->coef + b->coef;
                        if (sum) attach(sum, a->expon, &rear);
                        a = a->link;      b = b->link;
                        break;
                case 1:
                        attach(a->coef, a->expon, &rear);
                        a = a->link;
                        break;
                }
        }

        for (; a; a = a->link) attach(a->coef, a->expon, &rear);
        for (; b; b = b->link) attach(b->coef, b->expon, &rear);
        rear->link = NULL;
```

```
        temp = c; c = c->link; free(temp);
        return c;
}

void attach(int coefficient, int exponent, polyPointer* ptr)
{
        polyPointer temp;
        MALLOC(temp, sizeof(*temp));
        temp->coef = coefficient;
        temp->expon = exponent;
        (*ptr)->link = temp;
        *ptr = temp;
}

void erase(polyPointer* ptr)
{
        polyPointer temp;
        while (*ptr)
        {
                temp = *ptr;
                *ptr = (*ptr)->link;
                free(temp);
        }
}
```