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

//헤더노드를 가진 단일연결 환형리스트를 이용한 다항식 더하기 프로그램

#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define COMPARE(x, y) (((x)<(y)) ? -1 : ((x)==(y)) ? 0 : 1)
#define MALLOC(p, s) \
        if( !( (p) = malloc( s ) ) ){           \
                fprintf(stderr, "Insufficient memory");\
                exit(EXIT_FAILURE);\
        }

typedef struct polyNode* polyPointer;
typedef struct polyNode {
        int coef;
        int expon;
        polyPointer link;
}polyNode;

//헤더 노드를 가진 다항식(단일연결 환형리스트)
polyPointer a = NULL, b = NULL, c = NULL;
//다항식 a와 b에 대한 last포인터
polyPointer lastA, lastB;
//노드풀(체인)
polyPointer avail = NULL;

void erase(polyPointer* first);
polyPointer getNode(polyPointer* avail);
void retNode(polyPointer node, polyPointer* avail);
void cerase(polyPointer* ptr, polyPointer* avail);
polyPointer cpadd(polyPointer a, polyPointer b, polyPointer avail);
void attach(int coefficient, int exponent, polyPointer* ptr, polyPointer avail);
void insertFront(polyPointer* last, polyPointer node);
void insertLast(polyPointer* last, polyPointer node);
void printCList(polyPointer header);
void  inputPolyCL(char* filename, polyPointer* header, polyPointer* last, polyPointer
avail);
```

```
int main()
{
	polyPointer a = NULL, b = NULL, c = NULL;
	polyPointer lastA, lastB;

	polyPointer avail = NULL;


	inputPolyCL("a.txt", &a, &lastA, avail);
	inputPolyCL("b.txt", &b, &lastB, avail);

	//두 다항식의 정보를 출력
	printf("%9s", "a : ");		printCList(a);
	printf("%9s", "b : ");		printCList(b);

	//a+b의 결과를 c에 저장하는 다항식 더하기 수행
	c = cpadd(a, b, avail);
	//다항식 c를 출력
	printf("%9s", "a+b=c : "); printCList(c);

	//다항식a, b, c를 삭제
	cerase(&a, &avail);
	cerase(&b, &avail);
	cerase(&c, &avail);
	//printf("%9s", "avail : \n");		printCList(avail);

	//avail삭제
	erase(&avail);

	return 0;
}

//하나의 파일입력으로 "헤더노드를 가진 단일연결 환형리스트"로 구현된 하나의 다항식 생성
//제일 먼저 헤더 노드만으로 구성된 제로 다항식을 생성한 후, 파일 입력으로부터 하나씩 노드를 생성 한 후 리스트에 추가
void inputPolyCL(char* filename, polyPointer* header, polyPointer* last, polyPointer avail)
{
	polyPointer node;
	char order;					// a: ascending order(오름차순), d: descending order (내림차순)
```

```
FILE* fp;
int tp;

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

//헤더 노드만으로 구성된 제로 다항식 생성
*header = getNode(&avail);
(*header)->expon = -1;
*last = NULL;


fscanf_s(fp, "%c", &order, (int)sizeof(order));

//파일 입력형식이 'd'이면 insertLastCL 함수-리스트의 마지막 노드로 추가
//'a'이면 insertFrontCL 함수- 리스트의 첫 노드로 추가
if (order == 'd')
{
	insertLast(last, *header);
	node = getNode(&avail);

	while (!feof(fp))
	{
		tp = fscanf_s(fp, "%d%d", &node->coef, &node->expon);
		if (tp > 0)
		{
			insertLast(last, node);
			node = getNode(&avail);
		}

	}
	retNode(node, &avail);
}
// 'a' 오름차순 각 노드는 환형리스트의 첫 노드로 삽입
else
{
	insertFront(last, *header);
	node = getNode(&avail);
```

```
		while (!feof(fp))
		{
			tp = fscanf_s(fp, "%d%d", &node->coef, &node->expon);
			if (tp > 0)
			{
				insertFront(last, node);
				node = getNode(&avail);
			}

		}
		retNode(node, &avail);
	}

	fclose(fp);
}

//오름차순->각 노드는 환형리스트의 첫 노드로 삽입
void insertFront(polyPointer* last, polyPointer node)
{
	//마지막 노드를 가리키는 last포인터가 null일때
	if (!(*last))
	{
		*last = node;
		node->link = node;
	}
	//마지막 노드를 가리키는 last포인터가 null이 아닐때
	else
	{
		node->link = (*last)->link->link;
		(*last)->link->link = node;

		//첫 노드 삽입 시 lastA를 변경하고 이후는 lastA의 변경 없음
		if ((*last)->expon == -1)
			*last = node;
	}
}

//내림차순->각 노드는 환형리스트의 마지막 노드로 추가
void insertLast(polyPointer* last, polyPointer node)
{
	//마지막 노드를 가리키는 last포인터가 null일때
	if (!(*last))
	{
```

```
                *last = node;
                node->link = node;
        }
        //마지막 노드를 가리키는 last포인터가 null이 아닐때
        else
        {
                node->link = (*last)->link; //last가 가리키는 마지막 노드 뒤에 삽입
                (*last)->link = node; //last가 가리키는 마지막 노드 뒤에 삽입
                *last = node; //last포인터는 node를 가리킴
        }
}

//ptr이 가리키는 체인을 삭제
//체인이 처음부터 따라가며 모든 노드에 대한 메모리해제를 수행
void erase(polyPointer* ptr)
{
        polyPointer temp;
        //ptr이 비어있지 않을 때
        //temp는 ptr을 가리킴-> ptr을 이동-> temp에 있는 node를 free
        while (*ptr)
        {
                temp = *ptr;
                *ptr = (*ptr)->link;
                free(temp);
        }
        return;
}

//체인으로 구현된 노드풀로부터 노드를 하나 가져온다
polyPointer getNode(polyPointer* avail)
{
        polyPointer node;
        if (*avail)
        {
                node = *avail;
                *avail = (*avail)->link;
        }
        else
                MALLOC(node, sizeof(*node));
        return node;

}
```

```
//더 이상 사용하지 않는 노드 하나를 노드풀에 반환
void retNode(polyPointer node, polyPointer* avail)
{
        node->link = *avail;
        *avail = node;
}

void cerase(polyPointer* ptr, polyPointer* avail)
{
        polyPointer temp;
        if (*ptr)
        {
                temp = (*ptr)->link;
                (*ptr)->link = *avail;
                *avail = temp;
                *ptr = NULL;
        }
}

//a+b의 결과를 c에 저장하는 다항식 더하기 수행
//"헤더노드를 가진 단일 연결 환형리스트"로 표현된 두 다항식 a, b에 대해 더하기 연산을 수행하여 그 결과 다항식을 반환
polyPointer cpadd(polyPointer a, polyPointer b, polyPointer avail)
{
        polyPointer startA, c, lastC;
        int sum, done = FALSE;
        startA = a;
        a = a->link;
        b = b->link;
        c = getNode(&avail);
        c->expon = -1; lastC = c;

        do
        {
                switch (COMPARE(a->expon, b->expon))
                {
                case -1: /* a->expon < b->expon */
                        attach(b->coef, b->expon, &lastC, avail);
                        b = b->link;
                        break;

                case 0: /* a->expon = b->expon */
                        if (startA == a)
```

```
                        {
                                done = TRUE;
                        }
                        else
                        {
                                sum = a->coef + b->coef;
                                if (sum) attach(sum, a->expon, &lastC, avail);
                                a = a->link;        b = b->link;
                        }
                        break;

                case 1: /* a->expon > b->expon */
                        attach(a->coef, a->expon, &lastC, avail);
                        a = a->link;
                }
        } while (!done);

        lastC->link = c;

        return c;
}

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

//a, b 두 다항식의 정보를 출력
void printCList(polyPointer header)
{
        polyPointer temp;

        if (header)
        {
                temp = header->link;
                //printf("%+4dx^%d ", header->coef, header->expon);
                while (temp != header)
                {
```

```
                        if (temp->expon != 0) {
                                printf(" %+4dx^%d ", temp->coef, temp->expon);
                                temp = temp->link;
                        }
                        else
                        {
                                printf(" %+4d", temp->coef);
                                temp = temp->link;
                        }

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