자료구조
기말고사 프로젝트

제출일	21.06.11	**전 공**	컴퓨터소프트웨어공학과
과 목	자료구조 1	**학 번**	20184612
담당교수	홍 민 교수님	**이 름**	김동민

| 목 차 |

1. 단순 연결리스트의 곱셈 연산 프로그램

단순 연결리스트의 곱셈 연산 프로그램

- **교재 212페이지 프로그램 6.9에 있는 단순 연결리스트를 이용하여 data.txt 파일에 저장되어 있는 두 개의 다항식을 읽어서 이에 대한 곱셈 연산을 구현한다.**

1.1 문제 분석

조건 1 부호가 중복되어 출력되지 않도록 프로그램 수정이 필요하다.
조건 2 맨 뒤에 부호가 출력되지 않도록 프로그램 수정이 필요하다.
조건 3 data파일에는 다항식의 이름과 밑, 차수 순으로 저장되어있다.
조건 4 곱 연산을 수행했을 때 밑이 0이 나오는 경우 곱 리스트에서 제거하도록 한다.

- 이 문제는 data에 식의 이름과 계수와 지수의 값을 가지고 있다. 먼저 데이터로부터 식의 이름을 입력받고, 다항식의 내용을 입력받고 연결리스트에 값을 순서대로 저장한다.
연결리스트에 저장이 완료되었다면, 연결리스트를 출력하는데, 만약 계수의 부호가 -라면 부호가 중복으로 출력될 수도 있는 문제가 있다. 그러므로 부호가 +일 때와 -일 때를 구분하여 처리해야 한다. 또한 첫 번째 항 앞과 제일 마지막 항의 뒤에는 부호가 출력되면 안 되므로 이 부분을 처리하는 부분을 만들어야 할 것이다.
다항식의 곱을 수행할 때는 다항식1과 다항식2를 반복하며 밑은 다항식 밑의 곱을 저장하고 지수는 다항식 지수의 합을 저장한다. 그리고 만약에 같은 지수를 만나게 된다면 그 자리에서 계산을 수행한다. 만약 같은 지수를 끝까지 만나지 않는다면 뒤에 리스트를 추가한다.
곱 리스트가 입력이 완료되었다면 지수의 크기대로 내림차순 정렬을 해야하기 때문에, 계수와 지수를 구조체로 선언하여 정렬할 때 값의 이동을 쉽게 한다.

1.2 소스 코드

```c
/*
    학번 : 20134612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김홍민
    프로그램명 : 단순연결리스트3 응용 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning(disable:4996)

typedef struct{
    int coef;       //계
    int expon;      //지수
}element;
typedef struct ListNode{
    element data;                   //계와 지수 구조체
    struct ListNode *link;
}ListNode;
typedef struct ListType{
    int size;
    ListNode *head;                 //첫번째 가리키는 head포인터
    ListNode *tail;                 //맨뒤 가리키는 tail포인터
}ListType;

void error(char *message);              //에러메시지 출력 함수
ListType* create();                     //연결리스트 생성 함수
void insert_last(ListType* plist, int coef, int expon);         //리스트의 마지막에 리스트를 추가하는
void poly_print(ListType *plist, const char* formula);          //다항식을 출력하는 함수
void sort(ListType *plist);                                     //다항식을 내림차순 정렬하는 함수
void delete_zero(ListType* plist);                              //계수 지수에서 계수 0일때의 리스트를
void poly_mul(ListType* plist1, ListType* plist2, ListType* plist3);//곱셈계산 함수
void set_result(char *result, charpolyt, char *polyt2);         //결과 다항식의 이름을 지정하는 함

int main(){
    FILE *fp;
    ListType *list1, *list2, *list3;
    int tmp1, tmp2;
    char ch;
    char polyt1[80];
    char polyt2[80];
    char result[80];

    list1 = create();                   //다항식1생성
    list2 = create();                   //다항식2생성
    list3 = create();                   //다항식3생성

    fp=fopen("data.txt", "r");          //data를 읽기 전용으로 오픈
    if(fp==NULL){
        printf("파일이 열리지 않았습니다.₩n");
        return 0;
    }
    fscanf(fp, "%c", polyt1);           //data로부터 다항식 1의 이름 저장
    while(!feof(fp)){
        fscanf(fp, "%d%d", &tmp1, &tmp2);       //data로부터 임시변수1,2에 값을 읽어들
        fscanf(fp, "%c", &ch);                  //만약 ch가 줄바꿈자이면 처음 입력을 받는다는 신호이고
        insert_last(list1, tmp1, tmp2);         //tmp1값과 tmp2값을 연결에 다항식 리스트에 저장
        if(ch == '₩n')break;                    //ch가 줄바꿈 문자이면 반복문 종료
    }
    fscanf(fp, "%c", polyt2);           //data로부터 다항식 2의 이름 저장
    while(!feof(fp)){
        fscanf(fp, "%d%d", &tmp1, &tmp2);
        fscanf(fp, "%c", &ch);
        insert_last(list2, tmp1, tmp2);
        if(ch == '₩n')break;
    }
    poly_mul(list1, list2, list3);          //다항식3 곱셈 함수
    set_result(result, polyt, polyt2);      //polyt와 polyt2를 합하여 result문자열에 저장하는 함수

    poly_print(list1,  polyt);              //list1다항식 출력 다항식3 이름은 polyt에 저장
    poly_print(list2,  polyt2);             //list2다항식 출력 다항식3 이름은 polyt2에 저장
    poly_print(list3, result);              //list3다항식 출력 다항식3 이름은 result에 저장

    free(list1); free(list2); free(list3);
}
```

```c
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->data.coef = coef;                                  //temp에 값 저장
    temp->data.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_print(ListType *plist, const char* formula){
    ListNode *p = plist->head;

    printf("%s = ", formula);
    for(; p; p = p->link){
        if(p->data.coef>0){
            if(p!=plist->head) printf(" + ");
            if(p->data.expon==0)
                printf("%d", p->data.coef);
            else
                printf("%dx^%d", p->data.coef, p->data.expon);
        }
        else{
            if(p!=plist->head) printf(" - ");
            if(p->data.expon==0)
                printf("%d", -p->data.coef);
            else
                printf("%dx^%d", -p->data.coef, p->data.expon);
        }

    }
    printf("\n");
}
void sort(ListType *plist){
    ListNode *p, *q;
    element temp;
    for(p=plist->head; p!=NULL; p=p->link){
        for(q=p; q!=NULL; q=q->link){
            if(p->data.expon < q->data.expon){
                temp = p->data;
                p->data = q->data;
                q->data = temp;
            }
        }
    }
}
void delete_zero(ListType* plist){
    ListNode *p = plist->head;
    ListNode *remove;
    while(p!=NULL){
        if(p->link->data.coef == 0){
            remove = p->link;
            p->link = remove->link;
            free(remove);                                    //remove해제
        }
        p = p->link;
        if(p->link == NULL)break;
    }
}
void poly_mul(ListType* plist1, ListType* plist2, ListType* plist3){
    ListNode* a = plist1->head;
    ListNode* b = plist2->head;
    ListNode* c;
    while(a!=NULL){
        b=plist2->head;
        while(b!=NULL){
            c=plist3->head;
            while(c!=NULL){
                if(c->data.expon == a->data.expon + b->data.expon){
                    c->data.coef += a->data.coef * b->data.coef;
                    break;
                }
                c=c->link;
            }

            if(c==NULL) insert_last(plist3, a->data.coef*b->data.coef, a->data.expon + b->data.expon);

            b = b->link;
        }
        a = a->link;
    }
    delete_zero(plist3);
    sort(plist3);
}
void set_result(char *result, char*poly1, char *poly2){
    strcpy(result, poly1);
    strcat(result, " * ");
    strcat(result, poly2);
}
```

1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    프로그램명 : 단순연결리스트의 곱셈 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning(disable:4996)

typedef struct{
    int coef;
    int expon;
}element;
typedef struct ListNode{
    element data;
    struct ListNode *link;
}ListNode;
typedef struct ListType{
    int size;
    ListNode *head;
    ListNode *tail;
}ListType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성한다.

2. 필요한 헤더를 포함한다.

3. 리스트에 들어갈 밑과 지수부분을 구조체로 묶어 선언한다.
이 부분을 구조체로 선언한 이유는 나중에 내림차순 정렬을 할 때 편하게 하기 위함이다.

4. 각 항을 하나의 노드로 표현한다. element요소에는 계수(coef)와 지수(exp)가 있으므로 element자료형으로 이를 선언하고, 다음 항을 가리키는 링크필드로 구성한다.

5. 하나의 연결리스트를 두 개의 포인터 head와 tail로 표현한다. head는 첫 번째 노드를 가리키고 tail은 마지막 노드를 나타낸다. 추가로 연결리스트에 들어있는 항목의 개수를 나타내는 size함수를 추가한다.
단순 연결리스트에서 tail이 없다면 매번 포인터를 끝까지 옮겨서 추가해야 한다. 마지막 노드를 가리키는 포인터가 있는 경우 효율적으로 추가할 수 있다.

```
void error(char *message);
ListType* create( );
void insert_last(ListType* plist, int coef, int expon);
void poly_print(ListType *plist, const char* formula);
void sort(ListType *plist);
void delete_zero(ListType* plist);
void poly_mul(ListType* plist1, ListType* plist2, ListType* plist3);
void get_result(char *result, char*poly1, char *poly2);
```

6. 필요한 함수들을 선언한다.

- error함수는 오류메시지를 출력하고 강제 종료하는 함수이다.
- insert_last는 연결리스트의 마지막에 리스트를 추가하는 함수이다.
- poly_print함수는 다항식 연결리스트를 출력하는 함수이다.
- sort함수는 다항식을 지수에 따라 내림차순 정렬하는 함수이다.
- delete_zero는 다항식에 계수가 0이 있을 때 이 부분을 제거하는 함수이다.
- poly_mul함수는 다항식의 곱셈을 수행하는 함수이다.
- get_result함수는 다항식1과 다항식2를 합쳐서 다항식1 * 다항식2의 형태로 나타내는 함수이다.

```
int main( ){
    FILE *fp;
    ListType *list1, *list2, *list3;
    int tmp1, tmp2;
    char ch;
    char poly1[20], poly2[20], result[20];

    list1 = create( );
    list2 = create( );
    list3 = create( );

    fp=fopen("data.txt", "r");
    if(fp==NULL){
        error("파일이 열리지 않습니다.₩n");
    }
    fscanf(fp, "%s", poly1);
    while(!feof(fp)){
        fscanf(fp, "%d%d", &tmp1, &tmp2);
        fscanf(fp, "%c", &ch);
        insert_last(list1, tmp1, tmp2);
        if(ch == '₩n')break;
    }
    fscanf(fp, "%s", poly2);
    while(!feof(fp)){
        fscanf(fp, "%d%d", &tmp1, &tmp2);
        fscanf(fp, "%c", &ch);
        insert_last(list2, tmp1, tmp2);
        if(ch == '₩n')break;
    }
    poly_mul(list1, list2, list3);
    get_result(result, poly1, poly2);

    poly_print(list1,  poly1);
    poly_print(list2,  poly2);
    poly_print(list3, result);

    free(list1); free(list2); free(list3);
    fclose(fp); return 0;
}
```

7. 필요한 변수들을 선언한다.

- fp는 파일포인터, list1, list2, list3은 다항식1,2와 곱셈결과 다항식
- tmp1과 tmp2는 파일로부터 값을 읽어올 때 사용하는 변수
- ch는 다항식이 끝까지 입력되었는지 판단하는 변수
- poly1, poly2, result 문자열은 각각 다항식의 이름을 나타내는 변수

8. list1, list2, list3를 리스트를 생성하는 create()함수를 통해 리스트의 헤더를 생산한다. create함수는 ListType*을 리턴하는 함수이다.
fp를 읽기전용으로 오픈 후 파일이 정상적으로 오픈되었는지 검사한다.
만약 파일이 열리지 않았다면 error함수를 호출하여 매개변수로 오류내용을 전달한다.

9. 파일로부터 식의 이름을 입력받는다.
%s를 이용하여 data로부터 식의 이름을 poly1에 저장한다.
식의 이름 뒤에는 다항식의 계수, 지수가 순서대로 저장되어 있다. 파일로부터 2개의 값을 tmp1과 tmp2에 입력받고 list1의 순서대로 삽입한다.

10. 2개 값을 입력받은 후 파일로부터 문자형으로 하나를 입력받는다. 만약 ch가 공백문자이면 뒤에 내용이 계속 있다는 뜻이고, 줄 바꿈 문자이면 다항식 입력이 끝났다는 의미이므로 반복을 종료한다.
list2 다항식의 내용도 위와 똑같이 파일로부터 입력받는다.

11. poly_mul함수를 호출하여 다항식의 곱셈을 구한다.
list1과 list2 다항식을 곱한 값을 list3에 저장한다.
get_result함수는 결과식의 문자열을 저장하는 함수이다.
poly1의 이름 * poly2의 이름의 형태를 result문자열에 저장한다.

12. poly_print함수는 다항식을 출력하는 함수이다.
매개변수로 다항식리스트와 다항식의 이름을 전달한다.

13. main함수가 끝나면 동적할당한 list를 모두 메모리할당해제하고
fclose를 통해 파일을 닫는다.

```
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;
}
```

14. error함수는 오류 메시지를 출력하고 강제 종료하는 함수이다.
매개변수로는 메시지의 내용을 전달받고 함수에서 이 내용을 출력한다.
표준 출력장치 stdout를 사용하면 버퍼 문제로 출력이 제대로 되지 않을 수도 있다. stderr은 버퍼 없이 바로 출력한다. 그러므로 스트림에 출력하는 fprintf함수와 함께 사용하여, 문제가 발생할 경우 즉시 출력될 수 있는 장점이 있다. 오류가 출력되면 exit함수를 이용하여 강제 종료한다.

15. create함수는 리스트의 헤더를 생성하여 반환하는 함수이다. 반환형은 ListType*이다.
ListType을 동적할당으로 생성하여 head값과 tail값을 NULL로 설정하고 동적할당한 plist를 리턴한다.

```
void insert_last(ListType* plist, int coef, int expon){
    ListNode *temp = (ListNode*) malloc(sizeof(ListNode));
    if(temp==NULL) error("메모리 할당 에러");
    temp->data.coef = coef;
    temp->data.expon = expon;
    temp->link = NULL;
    if(plist->tail == NULL)
        plist->head = plist->tail = temp;
    else{
        plist->tail->link = temp;
        plist->tail = temp;
    }
    plist->size++;
}
```

16. insert_list함수는 리스트의 마지막에 추가로 삽입하는 함수이다.
plist는 연결리스트의 헤더를 가리키고, coef는 계수, expon은 지수이다.

17. 임시 노드 temp를 동적할당하여 메모리가 잘 할당되었는지 검사한다.
계수와 지수 값에 입력받은 값을 삽입하고 link는 NULL로 설정한다.

18. 만약 연결리스트의 테일 값이 NULL이라면 연결리스트에 노드가 하나도 없는 것이므로 맨 앞을 기리키는 헤드와 끝을 가리키는 테일에 temp노드를 삽입한다.

19. tail은 맨 끝 노드를 가리키므로 tail의 다음노드가 temp를 가리키도록 하고 그러면 temp가 끝 노드가 되므로 tail이 temp를 가리키도록 한다.
리스트의 개수가 증가하였으므로 리스트의 size를 1 증가시킨다.

```
void poly_print(ListType *plist, const char* formula){
    ListNode *p = plist->head;

    printf("%s = ", formula);
    for(; p; p = p->link){
        if(p->data.coef>0){
            if(p!=plist->head) printf(" + ");
            if(p->data.expon==0)
                printf("%d", p->data.coef);
            else
                printf("%dX^%d", p->data.coef, p->data.expon);
        }
        else{
            printf(" - ");
            if(p->data.expon==0)
                printf("%d", -p->data.coef);
            else
                printf("%dX^%d", -p->data.coef, p->data.expon);
        }

    }
    printf("₩n");
}
```

20. poly_print함수는 다항식을 출력하는 함수이다.
매개변수로는 연결리스트의 헤더와 다항식의 이름을 전달받는다. 여기서 식의 이름이 바뀌는 것을 방지하기 위해 const를 통해 상수로 전달받는다.

21. p에 plist의 head주소를 저장하고 먼저 다항식의 이름을 출력한다.
다항식의 계수에는 양수와 음수가 저장되어 있기 때문에 이를 구분하여 출력해야하고, 첫 항의 앞과 가장 끝항의 뒤는 연산자가 출력되면 안 되므로 이 부분도 구분해야 한다.

22. p를 다음 노드로 이동시키며 NULL이 될 때까지 반복한다.
만약 항의 계수가 양수일 때 p가 head가 아니라면 +를 출력한다.
p가 head와 같을 때 +를 출력한다면 첫 항 앞에 +가 출력되게 된다.


만약 지수가 0이라면 가장 끝항인 상수를 나타내므로 계수 값만 출력한다.
0이 아니라면 계수X^지수의 형태로 출력한다.

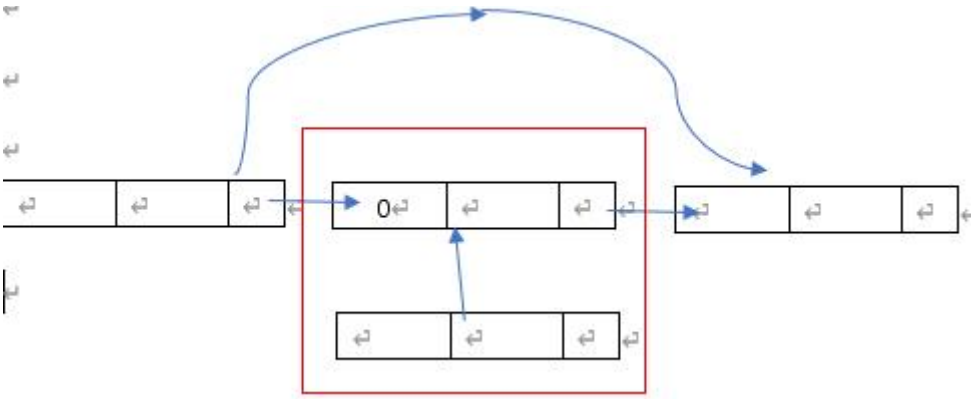
23. 똑같은 알고리즘으로 계수가 음수일 때도 -를 출력한다.
하지만 만약 -가 첫 항에 있을 때는 제일 앞에 -가 출력이 되어야 하므로 조건문 없이 무조건 출력한다.
계수의 값이 -일 때 -가 출력되면 안 되므로 값을 아예 바꾸지 않고 출력할 때만 -coef의 형태로 양수로 출력한다.
양수를 출력할 때와 같이 지수가 0일 때와 아닐 때를 구분하여 출력한다.

```
void sort(ListType *plist){
    ListNode *p, *q;
    element temp;
    for(p=plist->head; p!=NULL; p=p->link){
        for(q=p; q!=NULL; q=q->link){
            if(p->data.expon < q->data.expon){
                temp = q->data;
                q->data = p->data;
                p->data = temp;
            }
        }
    }
}
void delete_zero(ListType* plist){
    ListNode *p = plist->head;
    ListNode *remove;
    while(p!=NULL){
        if(p->link->data.coef == 0){
            remove = p->link;
            p->link = remove->link;
            plist->size--;
            free(remove);
        }
        p = p->link;
        if(p->link == NULL)break;
    }
}
```

24. sort함수는 다항식 리스트를 내림차순 정렬하는 함수이다.
이 부분을 위해 계수와 지수부분을 구조체 element로 선언하였다.
p를 head부터 NULL까지 반복하고 이중포문으로 q를 p부터 NULL까지 반복한다. 반복 도중 p의 지수가 q의 지수보다 작다면 두 값을 바꾸는 알고리즘을 수행한다. 이 부분을 수행하면 다항식이 내림차순으로 정렬된다.

25. delete_zero함수는 다항식의 곱셈연산 수행 후 계수가 0이 되는 항을 제거하는 함수이다. p의 노드를 하나씩 이동하며 반복문을 돈다.

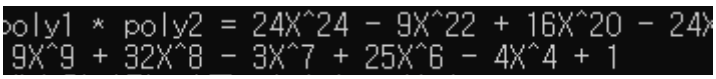
26. 만약 p의 다음노드의 계수가 0이라면 remove노드가 p의 다음 노드를 가리키도록 하고 p의 link는 remove의 링크를 가리키도록 한다.
리스트의 수가 줄었으므로 size는 --해주고 remove 노드를 삭제한다.
이를 그림으로 나타내면 이렇게 된다.



27. 만약 이 부분을 수행하지 않는다면 아래와 같이 결과 다항식의 계수에 0이 있는 것들이 포함되어 출력된다. 그러므로 이 부분은 반드시 필요하다.

* delete_zero를 호출 안 했을 때

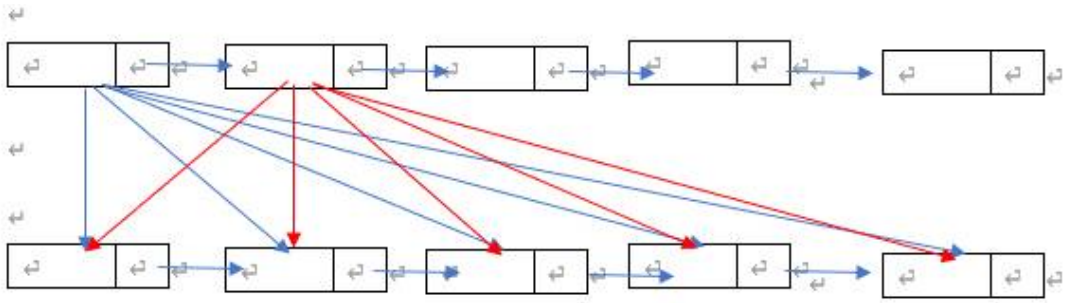
* delete_zero를 호출 했을 때



```
void poly_mul(ListType* plist1, ListType* plist2, ListType* plist3){
    ListNode* a = plist1->head;
    ListNode* b = plist2->head;
    ListNode* c;
    while(a!=NULL){
        b=plist2->head;
        while(b!=NULL){
            c=plist3->head;
            while(c!=NULL){
                if(c->data.expon == a->data.expon + b->data.expon){
                    c->data.coef += a->data.coef * b->data.coef;
                    break;
                }
                c=c->link;
            }
            if(c==NULL) insert_last(plist3, a->data.coef*b->data.coef, a->data.expon + b->data.expon);
            b = b->link;
        }
        a = a->link;
    }
    delete_zero(plist3);
    sort(plist3);
}
```

28. poly_mul은 list1식과 list2식의 곱을 list3에 저장하는 함수이다.
ListNode* 자료형으로 a, b, c를 선언한다. a는 list1의 헤드를 저장하고, b는 list2의 헤드를 저장하고, c는 list3의 헤드를 저장한다.

29. 먼저 a의 노드를 하나씩 이동하는 while문을 만들고, 그 안에 b의 노드를 하나씩 이동하는 while문을 만든다. 그리고 list3다항식에 계수는 a계수와 b계수의 곱을 삽입하고, 지수는 a지수와 b지수의 합을 삽입한다.
그러면 밑의 그림과 같은 알고리즘이 된다.



30. 그런데 만약 삽입도중에 a지수와 b지수의 합이 list3과 같은 지수를 가지면 그 자리에서 계산을 해야 한다.
그러므로 c의 노드를 하나씩 이동시키는 while문을 하나 더 만들고 c의 head부터 끝까지 노드를 이동시키며 지수가 같은 부분이 있는지 확인한다.

31. 만약 지수가 같은 부분이 없다면 list3다항식에 없는 지수이므로 다항식의 뒤에 삽입해야한다. 반대로 지수가 같은 부분이 있다면 그 자리에서 계산을 수행한 후 c의 반복문을 종료한다.

32. 지수가 있었는지 없었는지 확인하기 위해 c가 NULL값인지 확인한다.
c가 NULL값이라면 지수가 같은 부분이 없어서 끝까지 반복했다는 의미이므로 리스트 뒤에 삽입하는 함수를 호출한다.
반대로 지수가 있었다면 중간에 반복문을 종료했으므로 c가 NULL값이 아닐 것이다. 그러므로 리스트 뒤에 삽입을 하지 않는다.

```
void get_result(char *result, char*poly1, char *poly2){
    strcpy(result, poly1);
    strcat(result, " * ");
    strcat(result, poly2);
}
```

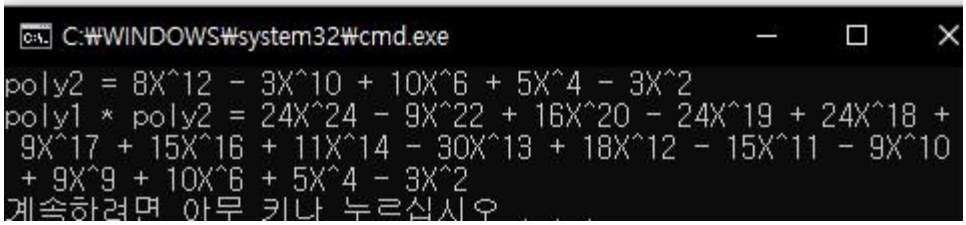
33. get_result함수는 결과 다항식의 이름을 만드는 함수이다.
다항식의 출력함수에서 문자열을 전달하고, 그 문자열을 출력하므로 결과식에 대한 문자열 또한 필요하다.

34. 먼저 result문자열에 strcpy로 poly1다항식의 이름을 복사한다.
strcat함수는 두 개의 문자열을 붙이는 함수이다. result문자열에 strcat함수로 result식 뒤에 *를 덧붙인 후 poly2다항식의 이름도 붙인다.
그러면 result다항식의 이름은 poly1이름 * poly2이름의 형태로 저장된다.

1.4 실행 창

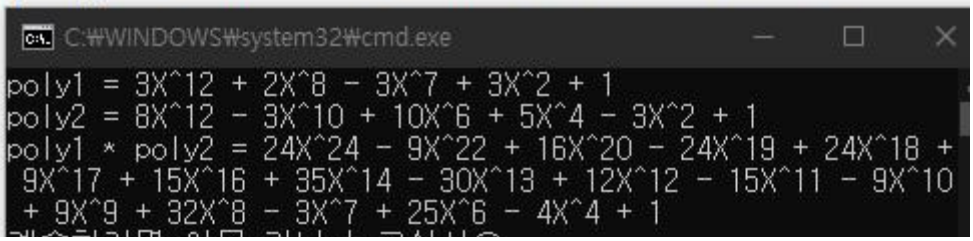
1. 예시 데이터파일과 동일

```
poly1 3 12 2 8 -3 7 1 0
poly2 8 12 -3 10 10 6 5 4 -3 2
```



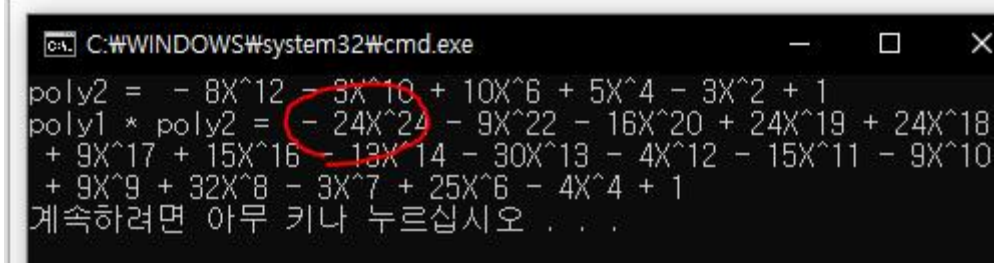
2. 곱셈 다항식 중간에 0이 나오는 경우
0인 부분 제거할 시

```
poly1 3 12 2 8 -3 7 3 2 1 0
poly2 8 12 -3 10 10 6 5 4 -3 2 1 0
```



3. 다항식의 첫 항이 음수일 경우

```
poly1 3 12 2 8 -3 7 3 2 1 0
poly2 -8 12 -3 10 10 6 5 4 -3 2 1 0
```



1.5 느낀 점

연결리스트는 자료구조실습시간에 잠깐 배웠지만 링크나 노드가 어떤 식으로 진행되는지 이해가 안 되는 부분이 조금 있었습니다. 하지만 이번 자료구조강의에서 연결리스트에 대해 배우면서 단순 연결리스트뿐만 아니라 더 나아가 원형 연결리스트, 이중 연결리스트에 대해서도 자세하게 배울 수 있었던 계기가 되었습니다.

가장 고민했던 부분 중 하나는 다항식을 출력하는 함수에서 연산자를 출력하는 위치를 정하는 부분이었습니다. 일단 이 부분을 해결하기 위해 계수가 양수일 때와 음수일 때를 구분하여 조건문을 작성하였습니다. 처음에는 양수일 때와 음수일 때 둘 다 첫 항 앞에 +나 -가 출력되지 않도록 했는데 첫 항의 계수가 음수인 경우 -가 출력되지 않는 경우가 발생해서 음수일 때는 -가 무조건 출력되도록 수정하였습니다.
또한 다항식의 곱셈을 수행할 때도 고민을 많이 했지만, 이 부분은 앞에서 했던 희소행렬의 곱셈을 할 때의 알고리즘과 많이 비슷한 부분이 있다고 생각해서 이와 비슷하게 프로그래밍 했던 것 같습니다. 예시에서 보여준 데이터파일은 곱셈 수행 시 계수가 0이 나올 때가 없었지만 제가 직접 0이 나오는 다항식을 만들어서 작성해보며 이를 토대로 0이 나오는 노드를 제거하는 함수 또한 만들 수 있었습니다.
책에 있는 코드를 그대로 작성하며 에러메시지를 출력하는 부분이 처음 접하는 코드라 이에 대해 찾아보았습니다. fprintf함수는 파일에 출력하는 함수인 줄로만 알고 있었지만 검색을 통해 스트림에 출력하는 함수라는 것을 알 수 있었습니다. 그리고 printf를 쓸 경우 버퍼 문제로 출력이 되지 않을 수도 있어서 stderr을 이용하여 버퍼 없이 바로 출력한다는 것을 알 수 있었습니다. 책에 있는 내용을 그대로 썼기 때문에 무심코 넘어갈 수도 있었지만 이렇게 쓴 이유를 찾아보니 코드가 더 잘 이해되었던 것 같습니다.

이번 기말프로젝트 1번 과제를 통해 연결리스트의 링크를 통해 노드를 연결하거나 정렬, 출력하는 방법에 대해 자세히 알 수 있는 기회가 되었습니다. 이를 계기로 앞으로 이를 응용하여 다른 문제도 풀어보고 싶다는 생각을 했습니다. 원형이나 이중연결리스트에 대해서는 문제가 나오지 않았지만 방학을 이용해 이 부분에 대해서도 복습해볼 것이라고 생각했습니다.

2. 원형 큐를 이용한 점수읽기 프로그램

원형 큐를 이용한 점수읽기 프로그램

- **교재 157 페이지 프로그램 5.2에 있는 배열로 구현된 원형 큐 프로그램을 이용하여 data.txt 파일에 저장되어 있는 이름, 학번, 국어, 영어, 수학 점수를 읽어서 동작되도록 구현한다.**

2.1 문제 분석

조건 1 data에는 enqueue 또는 dequeue가 저장되어 있다.
조건 2 enqueue가 있을 땐 이름, 학번, 국어, 영어, 점수가 저장되어 있다.
조건 3 원형 큐는 배열로 구현하고 data파일로부터 읽어온다.
조건 4 dequeue를 하면 삭제된 데이터를 출력한다.

- 이 문제는 data에 enqueue또는 dequeue가 저장되어있고, 만약 enqueue라면 그 뒤에 이름과 학번, 점수가 저장되어 있다. 따라서 큐의 요소는 구조체로 선언하며 안에 내용을 저장할 수 있는 변수들을 선언한다.
큐는 선입선출을 가지고 있는 구조로, 뒤에서 새로운 데이터가 추가되고 앞에서 데이터가 하나씩 삭제되는 구조를 가진다. 원형 큐는 나머지(modulo) 연산을 이용하여 인덱스를 원형으로 회전시키는 알고리즘이다.
큐는 요소를 가리키는 변수로 front와 rear를 가진다. 원형 큐를 초기화할 때, front와 rear값의 초기값을 0으로 설정한다. 삽입 시에는 rear를 먼저 증가하고 그 위치에 새로운 데이터를 삽입해야 하고, 삭제 시에도 front가 먼저 증가된 후 증가된 위치에서 데이터를 삭제한다.

2.2 소스 코드

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김종민
    프로그램명 : 원형 큐를 이용한 점수 읽기 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning(disable:4996)

#define MAX_QUEUE_SIZE 100                          //큐 사이즈 100
typedef struct{
    char *name;                                     //이름 저장 문자포인터
    int num;                                        //학번저장
    int kor, eng, math;                             //점수 저장 변수
}element;
typedef struct{
    element data[MAX_QUEUE_SIZE];                   //큐 구조체 배열
    int front, rear;                                //큐의 위치값
}QueueType;

void error(char *message);                          //에러메시지 출력함수
void init(QueueType *q);                            //큐 초기화 함수
int is_empty(QueueType *q);                         //큐 공백상태 확인함수
int is_full(QueueType *q);                          //큐 포화상태 확인함수
void queue_print(QueueType *q);                     //큐 출력함수
void enqueue(QueueType *q, element item);           //큐에 삽입하는 enqueue함수
element dequeue(QueueType *q);                      //큐를 삭제하는 dequeue함수
void input(FILE *fp,  QueueType*q);                 //enqueue시 필요한 함수의 모음
void output(QueueType *q);                          //dequeue시 필요한 함수의 모음

int main(){
    char str[20];
    QueueType q;
    FILE *fp;
    fp=fopen("data.txt", "r");                      //data.txt를 읽기전용으로 open
    if(fp==NULL){
        printf("파일이 열리지 않습니다.₩n");
        return 0;
    }
    init(&q);
    while(!feof(fp)){
        fscanf(fp, "%s", str);                      //str를 enqueue또는 dequeue
        if(strcmp(str,"enqueue")==0){               //만약 str이 enqueue라면 이뤄진
            input(fp,  &q);                         //enqueue할때 필요한 input함수 호출
        }
        else if(strcmp(str, "dequeue")==0){         //만약 str이 dequeue라면 이뤄진
            output(&q);                             //dequeue할때 필요한 output함수 호출
        }
        else{
            error("enqueue 또는 dequeue가 아닙니다.₩n");       //잘못된 입력이므로 에러메
        }
        queue_print(&q);
    }
    fclose(fp);
}
```

```
void error(char *message){                          //에러메시지 출력함수
    fprintf(stderr, "%s\n", message);               //stderr을 이용하여 버퍼없이 즉시 출력
    exit(1);
}
void init(QueueType *q){                            //큐 초기화 함수
    q->front = q->rear = 0;                         //원형큐이므로 front와 rear값을 0으로 설정
}
int is_empty(QueueType *q){                         //큐 공백상태 확인 함수
    return (q->front == q->rear);                   //front와 rear가 같다면 큐가 비어있는 상태이므로 같다면
}
int is_full(QueueType *q){                          //큐 포화상태 확인함수
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);    //rear에 1을 더하고 modulo연산 수행 후 이 값이 f
}
void queue_print(QueueType *q){                     //큐를 출력하는 함수
    int i = q->front;                               //i값은 front값으로 설정
    printf("QUEUE(front=%d rear=%d)\n", q->front, q->rear); //현재 front와 rear값 출력
    if(!is_empty(q)){
        do{                                         //do while문을 이용하여 한번은 반드시 출력하도록 설정
            i = (i+1) % MAX_QUEUE_SIZE;                 //원형큐이므로 i를 큐의 크기로 나머지연산 수행
            printf("%s | %d | %d | %d | %d |\n", q->data[i].name, q->data[i].num, q->data[i].kor,
                q->data[i].eng, q->data[i].math);       //큐 출력
            if(i==q->rear)break;                    //만약 i가 rear값과 같다면 큐의 내용이 끝났으므로 종료
        }while(i!=q->front);                        //i가 front값과 같아진다면 종료
    }
    printf("\n");
}
void enqueue(QueueType *q, element item){           //큐를 삽입하는 함수
    if(is_full(q))                                  //큐가 포화상태인지 확인
        error("큐가 포화상태입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;       //원형큐이므로 rear에 1을 더하고 나머지 연산수행후
    q->data[q->rear] = item;                        //rear위치에 item을 대입
}
element dequeue(QueueType *q){                      //큐를 삭제하는 함수
    if(is_empty(q))                                 //큐가 공백상태라면 큐를 삭제할 수 없으므로 오류
        error("큐가 공백상태입니다.\n");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;     //dequeue할땐 front값이 이동하므로 front를 나머지 연산
    return q->data[q->front];                       //front위치에 있는 data를 리턴
}
void input(FILE *fp,  QueueType*q){                 //data파일에서 enqueue가 입력되었을 때 수행하는 함수
    char s[20];
    element data;
    fscanf(fp, "%s%d%d%d%d", s, &data.num, &data.kor, &data.eng, &data.math);   //파일로부터 enqueue뒤에 있
    data.name=(char *)malloc(strlen(s)+1);                                      //data구조체에 있는 name은 포
    strcpy(data.name, s);                                                       //strcpy함수를 이용하여 내용
    enqueue(q, data);                                                           //enqueue수행
}
void output(QueueType *q){                          //data파일에서 dequeue가 입력되었을 때 수행하는 함수
    element data;
    data = dequeue(q);                              //리턴받은 data값을 받아옴
    printf("삭제된 데이터: %s %d %d %d %d\n", data.name, data.num, data.kor, data.eng, data.math);  //삭제된
    free(data.name);                                //name은 동적할당된 상태이므로 free수행
}
```

2.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    프로그램명 : 원형 큐를 이용한 점수 읽기 프로그램
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma warning(disable:4996)

#define MAX_QUEUE_SIZE 100
typedef struct{
    char *name;
    int num;
    int kor, eng, math;
}element;
typedef struct{
    element data[MAX_QUEUE_SIZE];
    int front, rear;
}QueueType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성한다.

2. 필요한 헤더를 포함한다.
큐의 최대 사이즈는 100으로 설정한다.

3. 큐에 들어갈 요소들을 구조체를 이용하여 선언한다.
이름을 저장하는 문자형포인터 name과 학번을 저장하는 num, 점수를 저장하는 kor, eng, math가 있다.

4. 구조체의 배열을 정의하고 삽입, 삭제를 위한 변수인 front와 rear를 만든다. 원형 큐에서 front는 큐의 첫 번째 요소의 하나 앞을 가리키고 rear는 큐의 마지막 요소를 가리킨다.
typedef를 이용하여 큐를 만들 때 선언문을 단순하게 사용할 수 있다.

```
void error(char *message);
void init(QueueType *q);
int is_empty(QueueType *q);
int is_full(QueueType *q);
void queue_print(QueueType *q);
void enqueue(QueueType *q, element item);
element dequeue(QueueType *q);
void input(FILE *fp,  QueueType*q);
void output(QueueType *q);
```

5. 필요한 함수들을 선언한다.

- error함수는 오류메시지를 출력하고 강제 종료하는 함수이다.
- init함수는 큐의 front와 rear를 초기화하는 함수이다.
- is_empty함수는 큐가 공백상태인지 확인하는 함수이다.
- is_full함수는 큐가 포화상태인지 확인하는 함수이다.
- queue_print함수는 원형 큐를 출력하는 함수이다.
- enqueue함수는 큐에 요소를 삽입하는 함수이다.
- dequeue함수는 큐에 있는 요소를 삭제하고 리턴하는 함수이다.
- input함수는 data파일에서 enqueue시에 파일을 읽어오는 함수이다.
- output함수는 data파일에서 dequeue시에 큐의 요소를 반환하는 함수이다.

```
int main(){
    char str[20];
    QueueType q;
    FILE *fp;
    fp=fopen("data.txt", "r");
    if(fp==NULL){
        printf("파일이 열리지 않습니다.₩n");
        return 0;
    }
    init(&q);

    while(!feof(fp)){
        fscanf(fp, "%s", str);
        if(strcmp(str,"enqueue")==0){
            input(fp,  &q);
        }
        else if(strcmp(str, "dequeue")==0){
            output(&q);
        }
        else{
            error("enqueue 또는 dequeue가 아닙니다.₩n");
        }
        queue_print(&q);
    }
    fclose(fp);
}
```

6. 필요한 변수들을 선언한다.
str은 파일에서 읽어올 때 enqueue, dequeue를 읽어오고, q는 큐를 선언하는 변수이다. 파일포인터 fp는 읽기 전용으로 오픈한다.

7. init함수를 이용해 큐를 초기화한다. q의 주소를 매개변수로 전달하여 함수에서의 내용을 공유할 수 있도록 한다.

8. data로부터 파일의 끝까지 입력받는다. 먼저 첫 내용을 %s를 이용하여 입력받는다. %s는 공백문자를 입력받지 못하기 때문에 str으로 입력받는 내용은 enqueue 또는 dequeue가 될 것이다.

9. str이 enqueue인지 dequeue인지 string.h헤더파일의 strcmp함수를 이용하여 검사한다. strcmp함수는 두 개의 문자열을 전달인자로 받아서 그 사전적 순서에 따라 대소를 비교하고 결과값을 리턴한다.
만약 두 개의 문자열이 같다면, strcmp함수는 0을 리턴한다.

10. str이 enqueue라면 data에는 이름, 학번, 점수등이 저장되어 있다.
파일로부터 큐에 데이터 입력을 받아야 하므로 input함수를 호출하여 파일포인터 fp와 q의 주소를 매개변수로 전달한다.

11. str이 dequeue라면 큐에서 삭제하고 삭제된 데이터를 출력해야하므로 dequeue함수와 그에 필요한 부분이 있는 output함수를 호출하여 q의 주소를 전달한다.

12. 만약 str이 enqueue도 dequeue도 아니라면 data가 잘못 입력된 것이므로 오류이므로 오류를 출력하는 error함수를 호출하고 강제로 종료한다.

13. enqueue나 dequeue를 했다면 큐를 화면에 출력하는 함수인 queue_print함수를 호출하여 현재 큐에 저장되어 있는 내용을 보여준다.

14. 파일이 끝에 도달했다면 fclose함수를 통해 파일을 닫는다.

```
void error(char *message){
    fprintf(stderr, "%s\n", message);
    exit(1);
}
void init(QueueType *q){
    q->front = q->rear = 0;
}
int is_empty(QueueType *q){
    return (q->front == q->rear);
}
int is_full(QueueType *q){
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}
```

15. error함수는 오류가 발생하면 이를 출력하는 함수이다.
매개변수로는 메시지의 내용을 전달받고 함수에서 이 내용을 출력한다.
표준 출력장치 stdout를 사용하면 버퍼 문제로 출력이 제대로 되지 않을 수도 있다. stderr은 버퍼 없이 바로 출력한다. 그러므로 스트림에 출력하는 fprintf함수와 함께 사용하여, 문제가 발생할 경우 즉시 출력될 수 있는 장점이 있다. 오류가 출력되면 exit함수를 이용하여 강제 종료한다.

16. init함수는 큐를 초기화하는 함수이다.
선형 큐에서는 front와 rear의 초기값을 -1로 설정하지만 원형 큐에서는 초기값을 0으로 설정한다. 또한 front는 큐의 첫 번째 요소의 하나 앞을 가리키고, rear는 마지막 요소를 가리킨다.

17. is_empty함수는 큐가 공백상태인지 확인하는 함수이다.
만약 큐의 front와 rear가 같다면 큐가 아무것도 없이 비어있는 상태이므로 이것이 참이면 1을 리턴하게 되고 만약 거짓이면 0을 리턴하게 된다.

18. is_full함수는 큐가 포화상태인지 확인하는 함수이다.
만약 front가 rear보다 하나 앞에 있으면 포화상태가 된다. 하지만 문제는 원형큐이기 때문에 rear에 1을 더하여 나머지 연산을 한다.
만약 이 값이 front와 같다면 1을 리턴하고 다르다면 0을 리턴한다.

```
void queue_print(QueueType *q){
    int i = q->front;
    printf("QUEUE(front=%d rear=%d)₩n", q->front, q->rear);
    if(!is_empty(q)){
        do{
            i = (i+1) % MAX_QUEUE_SIZE;
            printf("%s | %d | %d | %d | %d |₩n", q->data[i].name, q->data[i].num, q->data[i].kor,
                q->data[i].eng, q->data[i].math);
            if(i==q->rear)break;
        }while(i!=q->front);
    }
    printf("₩n");

}
```

19. queue_print함수는 현재 큐에 있는 내용을 출력하는 함수이다.
먼저 i에 큐의 front값을 저장한 후, 현재 front값과 rear값을 출력한다.

20. 큐가 공백상태인지 확인하는 is_empty함수를 호출하여 만약 공백상태가 아니라면 출력을 수행한다. (!is_empty)
do while반복문은 do다음에 오는 코드는 조건식과 상관없이 무조건 한번은 실행된다. do while반복문을 이용하여 i값이 큐의 front값과 같을 때까지 반복한다. 만약 중간에 i값이 rear값과 같아진다면 큐에 있는 내용이 모두 출력 된 것이므로 반복문을 종료한다.

21. 여기서 i값은 무조건 증가하는 것이 아닌, 나머지 연산을 이용하여 원형 큐에 맞게 값이 들어갈 수 있도록 한다. i에 1을 더한 후 큐의 크기로 나눈다면 원형으로 회전시키는 알고리즘을 구현할 수 있다.

```
void enqueue(QueueType *q, element item){
    if(is_full(q))
        error("큐가 포화상태입니다.");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}
element dequeue(QueueType *q){
    if(is_empty(q))
        error("큐가 공백상태입니다.₩n");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}
```

22. enqueue함수는 큐에 삽입하는 함수이다. 매개변수로 큐의 포인터와 요소의 내용을 전달받는다.
is_full함수를 호출하여 만약 큐가 포화상태라면 오류를 출력하는 error함수를 호출하여 오류메시지의 내용을 전달한다.

23. 큐는 나머지 연산자%를 이용하는 modulo연산을 사용한다.
먼저 삽입 전에 rear를 원형 회전시켜서 하나 증가시킨 후 증가된 위치에 데이터를 삽입한다. 무조건 하나씩 증가시킨다면 큐의 크기를 넘어가게 될 것이다. 그러므로 rear에 1을 더한 후 큐의 크기로 나머지연산을 한다.
결과적으로 만약 큐의 크기가 100이고 rear가 99라면 enqueue를 했을 때 rear가 0으로 원형 큐가 될 수 있다.
rear를 증가시켰다면 큐의 rear 위치에 입력받은 요소를 삽입해준다.

24. dequeue함수는 큐를 삭제하는 함수이다. 반환형을 element로 설정하여 삭제한 부분을 반환할 수 있도록 한다.
is_empty함수를 호출하여 큐가 공백상태인지 확인한다. 큐가 공백상태라면 error함수에 오류메시지를 전달하고 종료한다.

25. dequeue에서는 front를 움직여서 먼저 입력받은 내용을 먼저 삭제하는 선입선출의 내용을 만들 수 있다.
enqueue를 할 때와 마찬가지로 front에 front+1을 큐의 크기로 나머지 연산하여 modulo연산을 수행할 수 있도록 한다. front를 먼저 증가시킨 다음, 그 위치에 있는 큐의 데이터를 꺼내 와서 그 내용을 리턴한다.

```
void input(FILE *fp,  QueueType*q){
    char s[20];
    element data;
    fscanf(fp, "%s%d%d%d%d", s, &data.num, &data.kor, &data.eng, &data.math);
    data.name=(char *)malloc(strlen(s)+1);
    strcpy(data.name, s);
    enqueue(q, data);
}
void output(QueueType *q){
    element data;
    data = dequeue(q);
    printf("삭제된 데이터: %s %d %d %d %d₩n", data.name, data.num, data.kor, data.eng, data.math);
    free(data.name);
}
```

26. input함수는 data에서 enqueue를 입력받을 때 호출하는 함수이다.
매개변수는 파일포인터와 큐의 포인터를 전달받는다.
지금 파일포인터는 메인함수에서 enqueue를 읽은 상태이고 그 뒤엔 이름이 저장되어있다. 그러므로 문자형배열로 먼저 이를 입력받고 element요소를 가지는 data 변수를 선언하여 나머지를 저장한다.

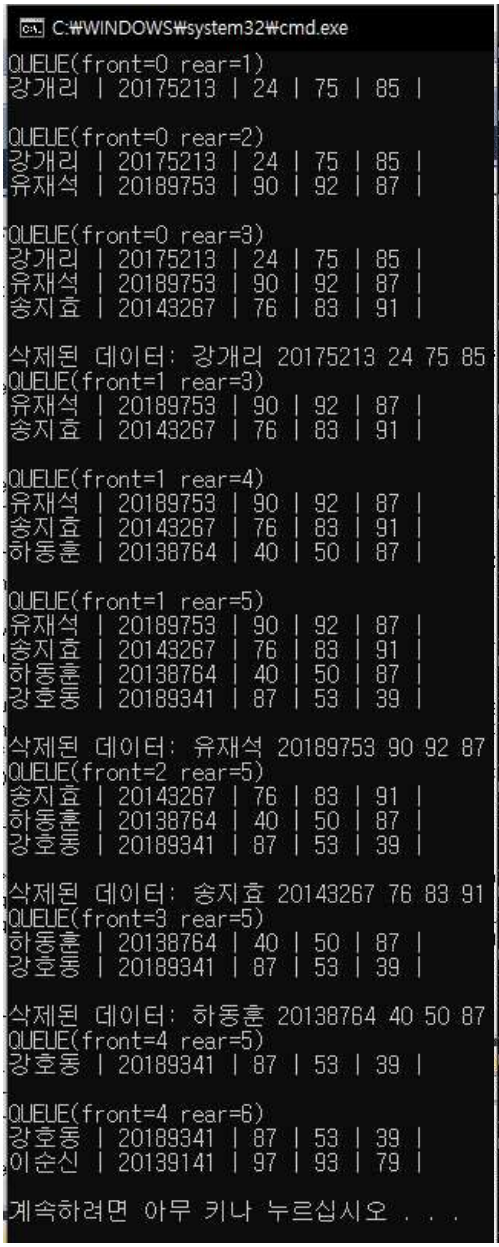
27. element구조체에서 이름을 선언할 때 차지하는 메모리를 아끼기 위해 포인터로 선언하였다. 하지만 포인터로는 바로 이름을 입력받을 수 없다. 따라서 element의 name포인터를 입력받은 문자형 배열의 길이+1만큼 동적할당한다. 여기서 +1을 한 이유는 문자열 뒤에 '\0'값을 추가하기 위함이다.

28. 동적할당이 완료되었다면 문자열을 복사하는 string.h헤더파일의 strcpy함수를 이용하여 name에 입력받은 문자열을 복사한다.
data 요소에 모든 값이 입력되었으므로 enqueue함수를 호출하여 data를 큐에 삽입한다.

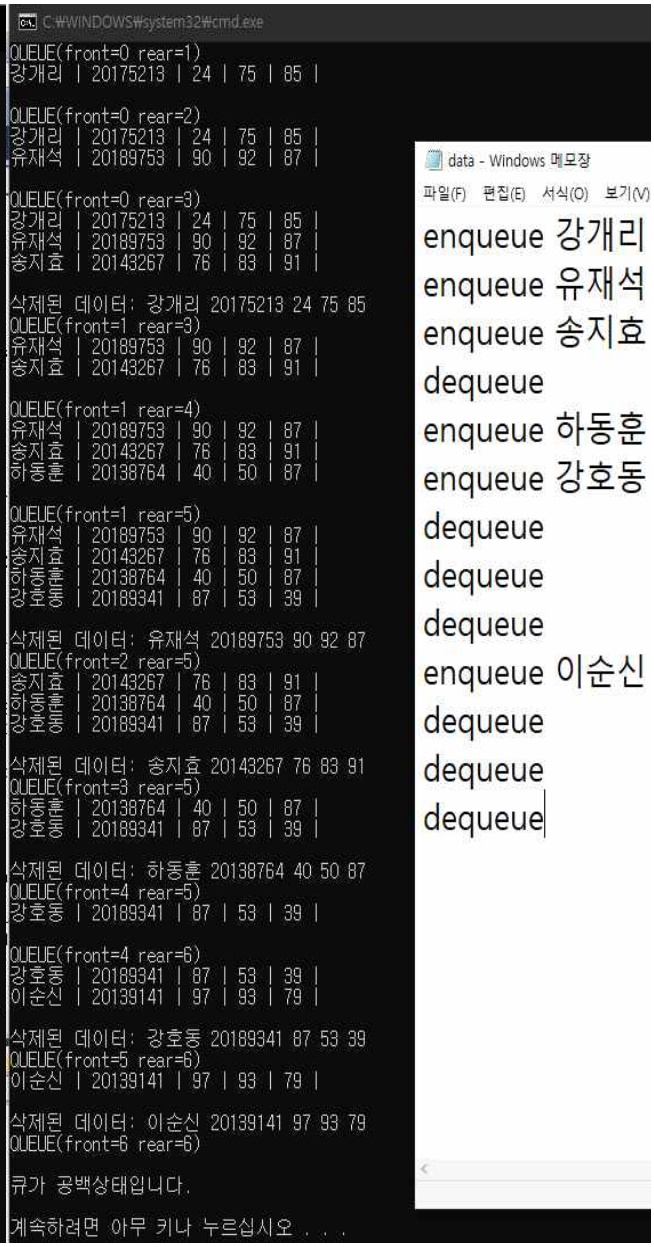
29. output함수는 data에서 dequeue를 입력받았을 때 호출하는 함수이다. 이 함수는 파일포인터는 쓰지 않기 때문에 큐의 포인터만 전달받는다.
dequeue는 요소 구조체를 반환하는 함수이므로 element자료형으로 선언한 data에 이를 전달받고, 삭제된 데이터(dequeue된 데이터)를 출력한다.
그리고 data의 name은 문자형 포인터로 동적할당된 상태이므로 free함수를 호출하여 동적할당한 메모리를 해제할 수 있도록 한다.

2.4 실행 창

1. 예시 데이터파일



2. 예시 데이터 파일+dequeue 3번 (큐가 공백상태일 때 체크)



2.5 느낀 점

기말고사의 두 번째 문제는 나머지 연산을 이용하여 원형 큐 배열을 구현하는 문제입니다. 강의시간에 이 부분에 대해 배울 때 나머지 연산에 대해서는 알고 있었지만 나머지 연산을 이용하여 무엇인가를 해본 것은 처음이었기 때문에 생소한 부분이 있기도 했습니다. 하지만 교수님께서 나머지 연산으로 원형 큐를 구현하는 방법을 line by line으로 설명해주셨기 때문에 보다 더 이해가 잘 되었던 것 같습니다.

이변 과제는 책에 있는 알고리즘을 가지고 부족한 부분을 추가하여 프로그래밍 해야 합니다. 저는 이 과제를 프로그래밍 할 때 무조건 책에 있는 내용을 따라 쓰는 것이 아니라 먼저 제가 코딩을 해보고 책에 있는 부분과 비교하며 책과 다른 부분이 있는지 확인하고 만약 다르다면 어떤 부분을 실수했는지 확인하며 프로그래밍 해보았습니다. 그렇게 하니 제가 부족하다고 생각했던 부분들을 채울 수 있는 기회가 되었습니다.
이번 프로젝트에서는 main함수를 최대한 간단하게 작성하려고 노력했던 것 같습니다. 입력받은 명령이 enqueue또는 dequeue일 때 main함수 안에서 이 부분을 처리하려고 하니 제가 보기에 가독성이 많이 떨어진다는 느낌을 많이 받았던 것 같습니다. 결국 프로그램은 main함수에서 작동하기 때문에 main함수를 가장 읽기 편하게 해야 한다고 생각했습니다. 그래서 enqueue일 때 필요한 요소를 모아놓은 input함수와 dequeue일 때 필요한 요소를 모아놓은 output함수를 만들고 main함수는 최대한 간결하고 가독성이 좋게 썼습니다.
또한 프로그램의 메모리를 줄이기 위해 이름을 포인터로 선언하여 파일로부터 입력받은 이름에 딱 맞게 이름을 저장할 수 있었습니다. 그 과정에서 문자열을 동적할당할 때 배열과는 약간 다르게 문자열의 길이에 '\0'문자를 포함하여 문자열의 길이+1으로 한다는 것을 알게 되었습니다. 이번 과제를 통해 문자열의 동적할당을 할 수 있는 방법 또한 배울 수 있었습니다.

이번 기말고사 과제를 통해 원형 큐에 대해 자세히 알 수 있었고 %을 응용하여 modulo연산을 하는 방법에 대해서도 배울 수 있는 계기가 되었습니다. 강의로만 들을 때 이해가 안 되었던 부분도 있었지만 다시 제가 일일이 프로그래밍해보며 큐라는 알고리즘을 다시 한 번 배울 수 있는 계기가 되었습니다. 이번 과제를 통해 만약 나중에 큐와 관련된 문제가 나온다면 자신 있게 풀 수 있을 것이라고 생각했습니다.

3. 기말 프로젝트를 하며 느낀 점

드디어 1학기가 끝났습니다. 엊그제가 3월 개학이었던 것 같은데 한 한기를 다시 돌아보니 길게만 느껴졌던 시간이 정말 빠르게 지나간 것 같습니다. 작년 군대 전역 후 고민이 정말 많았습니다. 개학 전 프로그래밍 공부를 미리 했지만 2년 동안의 공백이 생겼기 때문에 알고리즘을 이해할 수 있을지 생각이 많았습니다. 자료구조 수업을 하며 처음 보는 알고리즘도 있었고 이해가 안 되는 알고리즘도 있었지만 교수님께서 line by line으로 하나하나 설명해주신 덕분에 아직 부족한 저도 이해를 할 수 있었습니다.

처음에 강의를 들을 때는 과제가 너무 많아 힘들고 귀찮기도 했습니다. 하지만 수업시간에 배운 알고리즘을 가지고 프로그램을 작성하고 그것을 설명하는 레포트를 작성하고 설명을 적으며, 이 알고리즘이 왜 이런 순서로 진행되는지 깨닫고 교수님께서 말씀하신 내용을 다시 한 번 되새김질 할 수 있었던 계기가 되었습니다.
이번 기말과제는 자료구조 1강의의 마지막 레포트이다보니 지금까지 썼던 레포트 중에 가장 최선을 다했던 것 같습니다. 지금까지 나온 모든 과제들의 레포트는 최대한 라인하나하나 설명하며 쓰기위해 노력했고, 프로그래밍 또한 제가 알거나 배운 모든 함수나 알고리즘을 이용했습니다. 만약 모르는 부분이 있다면 제가 가지고 있는 책을 뒤지면서 찾으려고 노력했습니다. 그 덕분에 제가 놓치고 있었던 부분을 알게 될 때도 있었고, 배운 내용을 다시 공부할 수 있는 기회였습니다.

1학기는 끝났지만 2학기 전까지 2달이 넘는 방학이 있기 때문에, 프로그래밍 공부를 소홀히 하지 않고 앞으로 배울 내용을 미리 예습하여 2학기 때는 더 좋은 결과를 낼 수 있도록 노력하겠습니다. 자료구조 2에서는 더 좋은 모습 보일 수 있도록 하겠습니다.
한 학기동안 정말 고생 많으셨습니다. 감사합니다.

이상 “기말고사 프로젝트” 레포트를 마치겠습니다.
- 감사합니다. -