<데이터구조 및 실습>

프로그래밍 과제 #1

(다항식의 연산)

담당교수 : 이 상 호

제출일 : 2019.10.08

전공/학년 : 사이버보안 / 2

학번 : 1871082

이름 : 장 보 민

이메일 및 연락처 : qhals9443@ewhain.net

□ 문제 : 다항식의 연산

다항식을 Lecture 4. 45~49쪽의 헤더 노드가 있는 연결리스트로 표현하여 다음의 연산 들을 효율적으로 실행하는 프로그램을 작성하라.

=> 헤더 노드가 있는 연결리스트를 구현해야 하므로 연결리스트의 **노드**의 구조체와 연결리스트의 **헤더 노드**의 구조체를 따로 선언해야 할 것이다.

(1) 함수 poly_read() : 아래와 같은 형식의 다항식을 반드시 외부 파일로부터 읽어서 헤더 노드가 있는 단순 연결리스트로 만들고 그 다항식의 헤더 노드를 가리키는 포인터를 반환한다.

4 // p(x)의 0이 아닌 항의 개수

3 10

14 5 //p(x) = $3x^{10} + 14x^5 - 5x + 3$

-5 1

3 0

=> 외부 파일을 main()함수에서 읽어 들여와야 모든 함수에 FILE 포인터 변수 값을 넘겨주어서 값을 계속적으로 읽어올 수 있다. 읽어 들여온 후에 poly_read() 함수의 인자로 파일 포인터 변수 값을 넘겨주어야 poly_read()에서 파일을 다룰 수 있을 것이다. 헤더 노드가 있는 단순 연결리스트로 만들기 위해 헤더 노드를 가리키는 pointer값 또한 인자로 받아와야 한다. 어떤 헤더 노드를 기준으로 노드를 생성해 나가야 할 지 poly_read함수에서 알아야하기 때문이다. 또한 다항식이 헤더 노드를 가리키는 포인터를 반환해야 하므로 이 함수는

ListHeader* poly_read(ListHeader* plist, Fil* fp){} 형태가 될 것이다.

(2) 함수 poly_add() : 두 다항식 p(x)와 q(x)를 더하여 새로운 다항식 r(x)의 헤더 노드를 가리키는 포인터를 반환한다. 즉, $r(x) = p(x) + q(x)$이고 $r(x)$, $p(x)$, $q(x)$는 같은 표현 방식이어야 함.

=> 세 다항식의 헤더 노드를 가리키는 포인터는 main함수에서 선언을 하고, 각 포인터값들을 이 함수의 인자로 넘겨줘야 할 것이다. 또한 반환값이 헤더 노드를 가리키는 포인터 값이므로

이 함수는 **ListHeader* poly_add(ListHeader* plist1, ListHeader* plist2, ListHeader* plist3) {}** 형태가 될 것이다.

(3) 함수 poly_eval() : 주어진 정수값 x에 대하여 다항식 p(x)의 값을 계산하여 반환 한다. (Written Report #3의 3번 알고리즘 참조)

=> 주어진 정수값 x에 대하여 다항식의 값을 계산하여 반환해야 한다. 해당 다항식이 어떤 다항식인 지 알아야하기 때문에 이 함수의 인자로 헤더 노드를 기리키는 포인터 값을 전달해야하고, x 값을 전달해야 한다. 반환값은 int값이므로 함수의 형태는

int poly_eval(ListHeader* plist, int value){} 이렇게 될 것이다.

(4) 함수 poly_print() : 주어진 다항식 p(x)를 아래 예와 같이 출력한다. $P(x) = 3x^{10} + 14x^5 - 5x + 3$인 경우 p(x) = 3x^10 + 14x^5 + -5x^1 + 3x^0

=>각 다항식의 헤더 노드 포인터 값을 인자로 전달하고, 헤더 노드에 있는 정보(length, head, tail)를 이용하여 출력.

□ 문제 풀이 방법 및 알고리즘

(1) Poly_read()

```
ListHeader* poly_read(ListHeader* plist, FILE* fp)
{
        int length, coef, exp;
        fscanf(fp, "%d ", &length);
        plist->length = length;
        for (int i = 0; i < length; i++)
        {
                fscanf(fp, "%d %d ", &coef, &exp);
                insert_node_last(plist, coef, exp);
        }
        return plist;
}
```

다항식을 만들어 갈 노드의 기준이 되는 헤드 노드의 포인터와 파일 포인터를 인자로 전달받는 함수이다. Fscanf함수로 정수 하나를 불러와서 length변수에 저장한다. Plist가 저장하고 있는 주소, 즉, 헤드노드의 주소값에 접근해서 length값을 방금 읽어온 length으로 할당한다. Length값 만큼 for문을 반복하면서 다항식의 각 항에 해당하는 노드를 만들어낸다. 이 때 for문을 한 번 돌 때 마다 파일에서 정수 두 개를 불러와서 coef, exp 값에 할당하는 것과 더불어 insert_node_last() 함수를 호출하여 노드를 생성한다.

(2) Poly_add()

세 다항식의 헤드 노드의 포인터를 인자로 받는 함수이다. 헤드 노드를 사용하여 함수 내에서 ListNode type의 변수 a,b(노드)를 만들고 이 노드들로 계산을 수행할 것이다. while문에서 a와 b가 둘 다 NULL이 아닐 때 while문을 계속 수행한다. While문 내에는 a와 b의 exp를 비교하는 if문들이 존재한다. 만약 a와 b의 exp이 같을 때, sum(합)은 a의 coef와 b의 coef일 것이다.

이 결과값으로 $r(x)$를 만들어야 하기 때문에 insert_node_last()함수를 호출한다. 이 때 $r(x)$의 헤드 노드 포인터 plist3와, 계수가 될 sum, 차수 값인 a->exp값을 인자로 보낸다.

A의 exp가 b의 exp보다 높을 때는, insert_node_last의 인자로 plist3, a->coef, a->exp를 보낸다. B의 exp가 a의 exp보다 높을 때도 이와 같은 원리로 작동한다.

이 while문을 반복하면서 p3의 length를 세기 위해 매 while문마다 p3의 length를 1씩 증가시킨다.

만약 a나 b중 하나가 먼저 끝나게 되면 남아있는 항들을 모두 결과 다항식으로 복사하는 for문을 작성하였다. B가 먼저 끝났을 때, a의 나머지를 모두 복사하는 for문은 insert_node_last의 인자

로 plist3, a->coef, a->exp를 보낸다. 이 for문에서도 마찬가지로 p3의 length를 1씩 증가시켜줘야 한다. 다항식을 다 계산하고 나서는 plist3->length에 p3의 length를 대입해주고, plist3값을 return 한다.

```
ListHeader* poly_add(ListHeader* plist1, ListHeader* plist2, ListHeader* plist3)
{
	ListNode* a = plist1->head;
	ListNode* b = plist2->head;
	int sum;
	int p3_length = 0;
	while (a && b)
	{
		if (a->exp == b->exp) {
			sum = a->coef + b->coef;
			insert_node_last(plist3, sum, a->exp);
			a = a->link; b = b->link;
		}
		else if (a->exp > b->exp) {
			insert_node_last(plist3, a->coef, a->exp);
			a = a->link;
		}
		else
		{
			insert_node_last(plist3, b->coef, b->exp);
			b = b->link;
		}
		p3_length++;

	}
	for (; a != NULL; a = a->link)
	{
		insert_node_last(plist3, a->coef, a->exp); p3_length++;
	}
	for (; b != NULL; b = b->link)
	{
		insert_node_last(plist3, b->coef, b->exp); p3_length++;
	}
	plist3->length = p3_length;
	return plist3;
}
```

(3) Poly_eval()

헤드 노드를 가리키는 포인터와 value 값(지정된 x값)을 인자로 받는 pol_eval()함수이다. 전달받은 plist의 head값을 ListNode 포인터값 p에 대입하고, length, result 변수들에 값을 할당한 다음 while문을 반복한다. while문은 p값(노드를 가리키는 포인터값)이 NULL일 때까지 while문을 반복한다. while문을 돌 때마다 xpower를 1으로 초기화하고, p가 가리키는 노드의 exp값 만큼 for문을 돌아서 xpower를 계산한다. 각 항을 계산한 결과값(for문을 다 돌고 난 xpower값 * p의 coef)을 더해서 result 값을 갱신한다. 그리고 while문을 돌 때마다 p 값 또한 갱신해줘야 한다.

최종적으로 result값을 return한다.

```
int poly_eval(ListHeader* plist, int value)
{
	ListNode* p = plist->head;
	int length = plist->length;
	int result = 0;
	while (p) {
		int xpower = 1;
		for (int i = 0; i < p->exp; i++)
		{
			xpower *= value;
		}
		result = result + p->coef * xpower;.
		p = p->link;
	}
	return result;
}
```

(4) Poly_print()

각 리스트의 헤드노드의 포인터를 인자로 전달받는 함수이다.

ListNode 포인터 변수를 선언하여 이것으로 ploynomial을 프린트 할 것이다.

바깥 for문을 통해 plist1,plist2,plist3 중 하나를 p값으로 선택하고, 안쪽 for문을 통해 선택된 p값을 다항식으로 프린트하려 했으나, 이중 for문으로 되면 공간복잡도가 높아지기 때문에 그렇게 하지 않았다. 따라서 각 다항식 마다 별개로 for문을 돌려주었다.

```
void poly_print(ListHeader* plist1, ListHeader* plist2, ListHeader* plist3)
{
	ListNode* p;
	int length;
	int plus = 0;
	p = plist1->head; length = plist1->length;
	printf("p(x) = ");
	for (; p; p = p->link)
	{
		plus++;
		if (plus != 1 && plus<=length) { printf("+ "); }
		printf("%dx^%d ", p->coef, p->exp);
	}
	printf("Wn");
}
```

(5) insert_node_last()

헤드 노드를 가리키는 포인터 plist와 coef, exp를 인자로 받는 함수이다. 이 함수에서는 temp라는 새로운 노드를 동적 할당해서 temp의 coef 멤버에 인자로 전달받은 coef값을 저장하고, temp의 exp 멤버에는 인자로 전달받은 exp 값을 넣어준다. 만약 plist의 tail == NULL 일 때, 즉, 노드가 하나도 없을 때는 헤드 노드의 head와 tail에 temp의 주소를 대입해준다.

만일 노드가 하나라도 있을 때는 plist의 tail값이 가리키는 노드의 link에 temp의 주소를 넣어줌으로써 마지막 노드에 temp 노드를 추가한다. 그리고 plist의 tail값도 temp의 주소값으로 바꿔준다.

```
void insert_node_last(ListHeader* plist, int coef, int exp)
{
        ListNode* temp = (ListNode*)malloc(sizeof(ListNode));
        if (temp == NULL) return -1;
        temp->coef = coef;
        temp->exp = exp;
        temp->link = NULL;
        if (plist->tail == NULL)
        {
                plist->head = plist->tail = temp;
        }
        else {
                plist->tail->link = temp;
                plist->tail = temp;
        }
}
```

☐ Program

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

//연결리스트의 노드의 구조
typedef struct ListNode {
        int coef;
        int exp;
        struct ListNode* link;
}ListNode;

//연결리스트 헤더
typedef struct ListHeader {
        int length;

        //리스트 노드의 수를 저장하는 calumn
        ListNode* head;
        ListNode* tail;
}ListHeader;

//초기화 함수
ListHeader* init(ListHeader* plist)
{
        plist->length = 0;
        plist->head = plist->tail = NULL;
        return plist;
}

//기존 노드의 끝에 노드를 넣는 함수
void insert_node_last(ListHeader* plist, int coef, int exp)
{
        ListNode* temp = (ListNode*)malloc(sizeof(ListNode));      //새로운 노드 temp 할당
        if (temp == NULL) return -1;

        //전달받은 값들로 새로운 노드를 만듦.
        temp->coef = coef;
        temp->exp = exp;
        temp->link = NULL;                              //제일 마지막 노드의 link이므로 NULL
        if (plist->tail == NULL)                           //노드가 하나도 없을 때.
        {
                plist->head = plist->tail = temp;
        }
        else {
                plist->tail->link = temp;    //plist의 tail 노드의 link값을 temp로 바꿔줌.
                plist->tail = temp;                        //마지막에 temp값을 붙여주는 것
        }
}
```

```
//list3 = list1+list2
ListHeader* poly_add(ListHeader* plist1, ListHeader* plist2, ListHeader* plist3)
{
	ListNode* a = plist1->head;		//첫번째리스트 노드를 가리키는 포인터 a
	ListNode* b = plist2->head		//두번째리스트 노드를 가리키는 포인터 b
	int sum;
	int p3_length = 0;			//p3의 length를 count할 변수
	while (a && b)				//a와 b가 NULL이 아닐 때
	{
		//a의 차수와 b의 차수가 같을 때
		if (a->exp == b->exp) {
			sum = a->coef + b->coef;
			if (sum != 0) insert_node_last(plist3, sum, a->exp);
			a = a->link; b = b->link;
		}

		//a의 차수가 b의 차수보다 높을 때
		else if (a->exp > b->exp) {
			insert_node_last(plist3, a->coef, a->exp);
			a = a->link;
		}

		//b의 차수가 a의 차수보다 높을 때
		else
		{
			insert_node_last(plist3, b->coef, b->exp);
			b = b->link;
		}
		p3_length++;
	}
	//a나 b 중의 하나가 먼저 끝나게 되면 남아있는 항들을 모두 결과 다항식으로 복사
	for (; a != NULL; a = a->link)
	{
		insert_node_last(plist3, a->coef, a->exp); p3_length++;
	}
	for (; b != NULL; b = b->link)
	{
		insert_node_last(plist3, b->coef, b->exp); p3_length++;
	}
	plist3->length = p3_length;
	return plist3;
}
//다항식을 읽어오는 함수. ListHeader point값과 FILE 포인터값을 인자로 가짐.
ListHeader* poly_read(ListHeader* plist, FILE* fp)
{
	int length, coef, exp;
	fscanf(fp, "%d ", &length);			//파일의 첫번째 값을 읽어와서
	plist->length = length;				//ListHeader의 length값에 할당

	//length값만큼 for문을 반복하면서 insert_node_last함수를 호출하여 노드를 생성
	for (int i = 0; i < length; i++)
	{
		fscanf(fp, "%d %d ", &coef, &exp);
		insert_node_last(plist, coef, exp);
	}
	return plist;					//plist의 헤더값을 return
}
```

```
//다항식의 값을 계산하는 함수
int poly_eval(ListHeader* plist, int value)
{
	ListNode* p = plist->head;		//전달받은 plist(ListHeader포인터값)의
					head값을 ListNode포인터값 p에 대입.

	//각 변수들에 값을 할당
	int length = plist->length;
	int result = 0;

	//p가 NULL일때까지 while문을 반복
	while (p) {
		int xpower = 1;

	//while문을 돌 때마다 xpower를 1로 초기화.
		//p가 가리키는 노드의 exp값만큼 for문을 돌아서 xpower를 계산함.
		for (int i = 0; i < p->exp; i++)
		{
			xpower *= value;
		}
		result = result + p->coef * xpower; //result에는 while문을 돌 때마다 '항을
계산한 결과값'이 더해짐.
		p = p->link;		//p가 다음 노드를 가리키도록 p값도 갱신해줌.
	}
	return result;			//계산한 결과값을 return
}
void poly_print(ListHeader* plist1, ListHeader* plist2, ListHeader* plist3)
{
	ListNode* p;			//ListNode 포인터변수 p를 선언
	int length;	//각 polynomial의 length를 저장할 int타입의 변수 length선언
	int plus = 0;	//plus를 출력할 때 도움을 줄 plus변수. 매번 갱신돼야함.

	//plist1에 대한 출력
	p = plist1->head; length = plist1->length;
	printf("p(x) = ");
	for (; p; p = p->link)
	{
		printf("%dx^%d ", p->coef, p->exp);
		plus++;
		if (length != plus) { printf("+ "); }
	}
	printf("₩n");


	//plist2에 대한 출력
	plus = 0;
	p = plist2->head; length = plist2->length;
	printf("q(x) = ");
	for (; p; p = p->link)
	{
		printf("%dx^%d ", p->coef, p->exp);
		plus++;
		if (length != plus) { printf("+ "); }
	}
	printf("₩n");
```

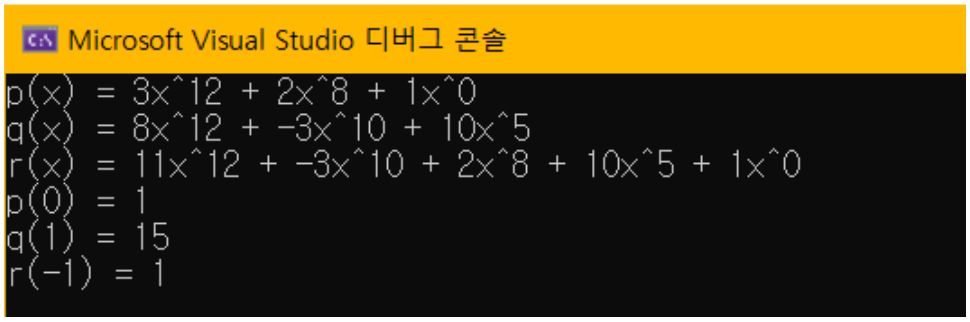
```
        //plist3에 대한 출력
        plus = 0;
        p = plist3->head; length = plist3->length;
        printf("r(x) = ");
        for (; p; p = p->link)
        {
                printf("%dx^%d ", p->coef, p->exp);
                plus++;
                if (length != plus) { printf("+ "); }

        }
        printf("Wn");

}
void main()
{
        ListHeader list1, list2, list3;                 //ListHeader선언.
        FILE* fp = fopen("poly1.txt", "r");             //읽어들여올 파일 선언,초기화

        //연결리스트의 초기화
        ListHeader* plist1 = init(&list1);
        ListHeader* plist2 = init(&list2);
        ListHeader* plist3 = init(&list3);

        //각 다항식의 헤더 노드를 가리키는 포인터(ListHeader 포인터)를 각 변수에 저장
        plist1 = poly_read(&list1, fp);
        plist2 = poly_read(&list2, fp);

        //두 다항식을 더하여 새로운 다항식 r(x)의 헤더 노드를 가리키는 포인터를 반환받아
저장.
        plist3 = poly_add(plist1, plist2, plist3);

        //각 다항식들을 출력
        poly_print(plist1, plist2, plist3);

        //x 값을 저장할 value, 계산 결과를 저장할 result 변수를 선언
        int value, result;

        fscanf(fp, "%d ", &value);                      //file에서 value값을 불러오고
        result = poly_eval(plist1, value);          //result에 다항식을 계산한 값을 저장
        printf("p(%d) = %dWn", value, result);          //값 출력

        fscanf(fp, "%d ", &value);                      //file에서 value값을 불러오고
        result = poly_eval(plist2, value);      //result에 다항식을 계산한 값을 저장
        printf("q(%d) = %dWn", value, result);          //값 출력

        fscanf(fp, "%d ", &value);                      //file에서 value값을 불러오고
        result = poly_eval(plist3, value);      //result에 다항식을 계산한 값을 저장
        printf("r(%d) = %dWn", value, result);          //값 츌력



        fclose(fp);                                     //파일 닫기

}
```

□ 입출력의 예

□입력 값에 따른 올바른 출력 값

입력값	3 3 12 2 8 1 0 3 8 12 -3 10 10 5 0 1 -1
출력값	$p(x) = 3x^{12} + 2x^{8} + 1x^{0}$
	$q(x) = 8x^{12} - 3x^{8} + 1x^{0}$
	$r(x) = 11x^{12} - 3x^{10} + 2x^{8} +$
	p(0) = 1
	q(1) = 15
	r(−1) = 1

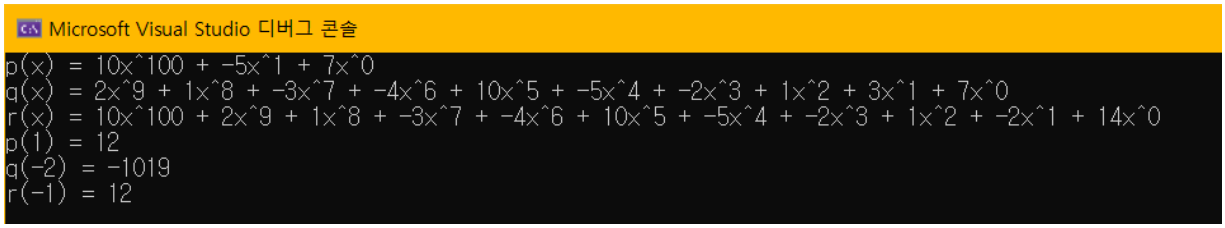
□실제 출력 결과



□입력 값에 따른 올바른 출력 값

입력값	310 100 -5 1 7 0 10 2 9 1 8 -3 7 -4 6 10 5 -5 4 -2 3 1 2 3 1 7 0 1 -2 -1
출력값	$p(x) = 10x^{100} - 5x^{1} + 7x^{0}$
	$q(x) = 2x^{9} + 1x^{8} - 3x^{7} - 4x^{6} + 10x^{5} - 5x^{4} - 2x^{3} + 1x^{2} + 3x^{1} + 7x^{0}$
	$r(x) = 10x^{100} + 2x^{9} + 1x^{8} - 3x^{7} - 4x^{6} + 10x^{5} - 5x^{4} - 2x^{3} + 1x^{2} - 2x^{1} + 14x^{0}$
	p(1) = 12
	q(−2) = −1019
	r(−1) = 12

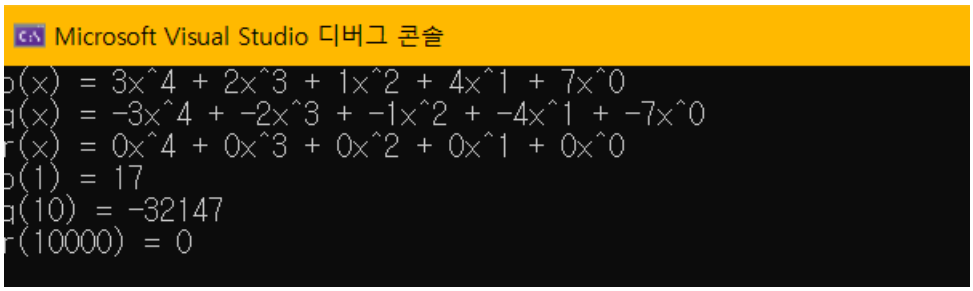
□실제 출력 결과



□입력 값에 따른 올바른 출력 값

입력값	5 3 4 2 3 1 2 4 1 7 0 5 -3 4 -2 3 -1 2 -4 1 -7 0 1 10 10000
출력값	$p(x) = 3x^4 + 2x^3 + 1x^2 + 4x^1 + 7x^0$
	$q(x) = -3x^4 - 2x^3 - 1x^2 - 4x^1 - 7x^0$
	$r(x) = 0x^4 + 0x^3 + 0x^2 + 0x^1 + 0x^0$
	p(1) = 17
	q(10) = -32147
	r(10000) = 0

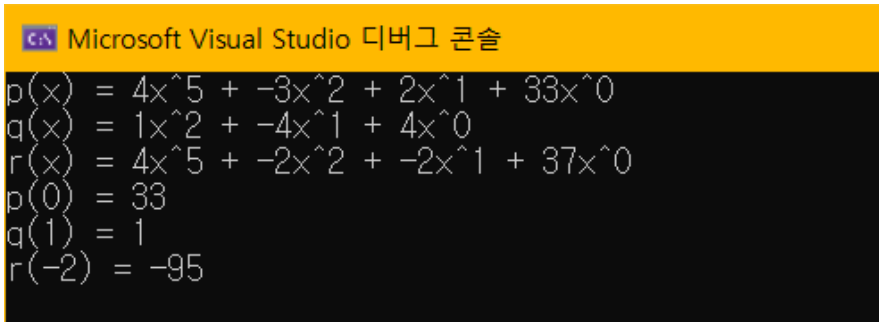
□실제 출력 결과



□입력 값에 따른 올바른 출력 값

입력값	4 4 5 -3 2 2 1 33 0 3 1 2 -4 1 4 0 0 1 -2
출력값	$p(x) = 4x^5 - 3x^2 + 2x^1 + 33x^0$
	$q(x) = 1x^2 - 4x^1 + 4x^0$
	$r(x) = 4x^5 - 2x^2 - 2x^1 + 37x^0$
	p(0) = 33
	q(1) = 1
	r(-2) = -95

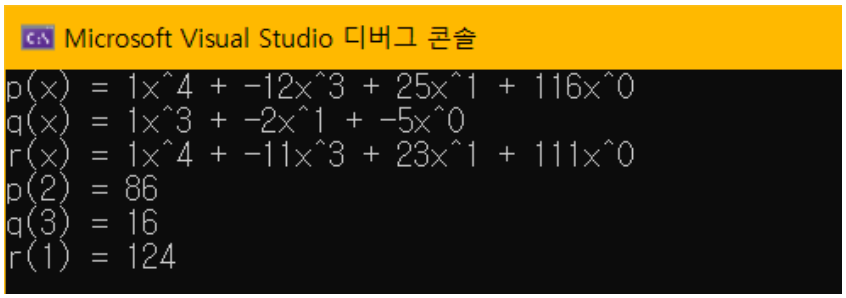
□실제 출력 결과



□입력 값에 따른 올바른 출력 값

입력값	4 1 4 -12 3 25 1 166 0 3 1 3 -2 1 -5 0 2 3 1
출력값	$p(x) = 1x^4 - 12x^3 + 25x^1 + 116x^0$
	$q(x) = 1x^3 - 2x^1 - 5x^0$
	$r(x) = 1x^4 - 11x^3 + 23x^1 + 111x^0$
	p(2) = 86
	q(3) = 16
	r(1) = 124

□실제 출력 결과



□ 결과 분석 및 토의

(1) Poly_read()

□ 다항식을 외부 파일로부터 읽어서 헤더 노드가 있는 단순 연결리스트로 만들고 그 다항식의 헤더 노드를 가리키는 포인터를 반환할 수 있었다.

□ 해당 함수 안에서는 for문을 length번 반복한다. 이 함수에서의 기본 연산을 fscanf로 한다면, 시간복잡도 T(n) = O(n)이 되고, 공간복잡도는 for문을 n번 반복할 때마다 n만큼의 시스템 스택이 필요하므로 S(n) = O(n)이 된다.

(2)poly_add()

□ 두 다학식 p(x)와 q(x)를 더하여 새로운 다항식 r(x)의 헤더 노드를 가리키는 노드를 반환할 수 있었다. r(x)=p(x)+ q(x)를 계산할 수 있었고, r(x),p(x),q(x)를 같은 표현 방식으로 나타낼 수 있었다.

□이 함수에서는 a와 b 둘 중 하나가 끝날 때 까지 while문을 반복하면서 if문의 경우에 따라 insert_node_last를 호출함으로써 덧셈연산을 수행한다. Insert_node_last를 기본 연산으로 두면 이 함수에서 시간복잡도는 T(n) = O(n)이 되고, 공간복잡도 S(n)은 O(n)이 된다.

(3)poly_eval()

□ 주어진 정수값 x에 대하여 다항식 p(x)의 값을 계산하여 반환할 수 있었다.

□ 이 함수에서는 p가 NULL이 NULL일 때까지 while문을 반복한다. While문 안에 또 for문이 있으므로 최악의 경우 시간 복잡도는 T(n) = O(n^2)이 되고, 공간복잡도 S(n)=O(n)이 된다.

(4)poly_print()

□ 주어진 다항식을 정해진 형태로 출력할 수 있었다.

□ 이 함수에서는 각 다항식의 ListHeader 포인터를 인자로 받고 그에 대한 연산을 for문으로 수행한다. 때문에 T(n) = O(n)이 되고, 공간복잡도 S(n) = O(n)이 된다.

위의 함수들을 통해 헤더 노드가 있는 단순 연결리스트를 표현할 수 있었다. 헤드 노드가 있으니, 삽입, 삭제 연산을 간단하게 구현할 수 있었고, list가 NULL인지 아닌 지 판단하는 것 또한 헤더 노드만으로 판단 가능했다. Poly_eval함수에서 시간 복잡도가 O(n^2)이 나왔다. 시간 복잡도를 줄여서 효율적으로 계산할 수 있는 방안을 생각해보았더니 while문을 굳이 쓰지 않고도 for문의 조건으로 p가 NULL이 아닐 때를 붙여주면 되는 방법이 있었다. 이 과제를 통해서 프로그래밍을 할 때, 시간복잡도와 공간복잡도에 대해서 생각하면서 코드를 짤 수 있었고, 효율적으로 연결리스트를 구현하는 방법을 알게 되었다. 앞으로도 공간복잡도와 시간복잡도를 잘 계산하여, 효율적인 함수를 구현해야 겠다는 생각을 하였다.