

자료구조2 실습 레포트



과목명		자료구조2 실습
담당교수		홍 민 교수님
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20204059
이름		이예빈

목 차

1. 파일 입출력, 동적 할당과 단순 연결리스트를 이용한 정수 버블 정렬 프로그램

1.1 문제 분석

1.2 소스 코드

1.3 소스 코드 분석

1.4 실행창

1.5 느낀점

2. 파일 입출력, 동적 할당과 단순 연결리스트, 정렬을 이용한 학생 성적 프로그램

2.1 문제 분석

2.2 소스 코드

2.3 소스 코드 분석

2.4 실행창

2.5 느낀점

3. 느낀점

1. 파일 입출력, 동적 할당과 단순 연결리스트를 이용하여 저장한 정수 자료들에 대한 버블 정렬 프로그램

■ 파일에서 자료 읽어오기

- 파일에서 자료를 읽어와 링크드 리스트에 저장하는 파일을 작성하고 아래 조건을 만족하게 출력하는 프로그램을 작성하라.
 - 파일 `data.txt`에 저장되어 있는 정수를 사용할 것
 - 링크드 리스트를 이용하여 저장
 - 동적 할당 이용
 - 리스트에 저장된 정수들을 오름차순으로 정렬하여 출력(버블 정렬)

```
C:\WINDOWS\system32\cmd.exe
<오름차순 정렬>
1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9
```

data.txt

파일(F) 편집(E) 서식(O) 보기(V)

4 5 9 7 2 1 6 8

1.1 문제 분석

이 문제는 파일에 저장되어 있는 정수 자료들을 읽어 동적 할당을 이용하여 연결 리스트에 저장하고, 리스트에 저장된 정수들을 오름차순으로 정렬하고 출력을 하는 문제이다. 문제를 해결하기 위해선 다음과 같이 크게 5 가지를 고려할 수 있다.

1. 구조체

- 연결 리스트로 구현하기 위하여, 데이터 필드와 링크 필드를 가지는 노드 구조체를 선언한다. 데이터 필드는 정수로, 링크 필드는 자체 참조 구조체 포인터를 사용하여 다음 노드를 가리킬 수 있는 포인터를 선언한다.

2. 파일 입출력

- 우선 파일을 열어 정수 자료들을 하나씩 읽어야 한다. 리스트에 저장하는 것이므로, 정수 데이터 한 개를 읽을 때마다 해당 데이터에 대한 메모리를 동적 할당하여 리스트에 삽입하도록 한다.

3. 동적 할당과 연결 리스트에 삽입

- 읽은 정수 데이터 하나에 대하여, 메모리를 동적 할당하고 리스트의 헤드에 첫 노드로 삽입한다. 헤드에 삽입하는 것이므로, 새로 삽입하는 노드의 링크 필드에는 기존의 head의 주소를 대입한 후 다시 head 포인터를 해당 노드의 주소로 변경해주면 리스트의 첫 노드로 삽입하기를 반복하게 된다. 이때 head 포인터의 값이 지속적으로 바뀌게

되므로 리스트에 삽입하는 함수(insert_last)에서는 head 포인터를 반환하여 호출한 쪽에서도 head 가 변경될 수 있도록 해야 한다.

- 리스트에 대한 메모리를 모두 사용하고 난 뒤에는, 리스트의 각 노드에 방문하며 해당 노드에 대한 메모리를 해제하도록 한다. 첫번째로 오는 노드부터 차례대로 메모리를 해제하도록 한다. 중간에 있는 어떤 노드에 대한 데이터를 삭제하는 경우가 아니므로, 전체 메모리를 해제하는 하나의 함수로 구현하였다.

4. 오름차순 버블 정렬

- 데이터 n 개에 대하여, 최악의 경우 최대 $(n-1)$ 번 반복하며 첫번째 데이터부터 순회하며 데이터와 다음 데이터의 값을 비교하여 다음 데이터의 값이 더 큰 경우 두 값을 교환하기를 반복하고, 각 반복마다 얻어지는 가장 큰 데이터 값을 가장 마지막으로 이동하게 하는 정렬의 방법이다.
- 한 번의 반복에 대하여, 가장 큰 데이터가 맨 끝으로 이동하게 되므로, 한 번의 순회가 마무리된 후 다음 순회 때에는 더 이상 이전의 마지막 데이터를 다루지 않아도 된다. 이를 위해서 각 순회마다 주어지는 가장 큰 정수를 가지게 되는 마지막 노드를 가리키는 포인터를 설정하여, 각 반복이 끝날 때마다 이 포인터가 차례대로 왼쪽 노드로 순차적으로 이동할 수 있도록 설정하였다.
- 또 고려해야할 점은, 최악의 상황이 아니며 어떠한 순회의 과정에서 더 이상 두 값의 교환이 일어나지 않는 경우이다. 이러한 경우, 그 이상의 필요하지 않는 반복을 하지 않도록 하는 것이 효율적이라고 생각했기 때문에 하나의 플래그 값을 설정하여, 일정 플래그 값일 때만 다음 순회를 지속할 수 있도록 한다.

5. 리스트 출력

- 오름차순 정렬된 리스트에 대한 단순 출력이므로, head 포인터가 가리키는 첫번째 노드부터 차례대로 정렬된 정수들을 출력하도록 한다.

1.2 소스 코드

```

1  /*
2  → 작성자: ·이예빈 (20204059)
3  → 작성일: ·2021.09.13
4  → 프로그램명: ·파일에서 정수 자료를 읽어, 동적으로 메모리를 할당 받아 연결 리스트에 저장하고,
5  → → → 리스트에 저장된 정수들을 버블 정렬을 사용하여 오름차순으로 정렬하여 출력한다.
6  */
7
8  #include <stdio.h>
9  #include <stdlib.h>
10
11  // 연결 리스트의 노드 구조체 선언
12  typedef int element;
13  typedef struct ListNode {
14  → element data;
15  → struct ListNode *link;
16  } ListNode;
17
18  // 함수 원형 선언
19  ListNode *insert_first(ListNode *head, element value);
20  void print_list(ListNode *head);
21  void bubble_sort(ListNode *head);
22  void clear(ListNode *head);
23
24  int main() {
25  → FILE *fp; → → → → → // 파일 포인터
26  → ListNode *list = NULL; → → // 연결 리스트의 헤드 포인터
27  → element tmp; → → → → // 데이터를 임시로 저장할 element형 변수 tmp
28
29  → // 파일 열기 (오류 코드 포함)
30  → fp = fopen("data.txt", "rt");
31  → if (fp == NULL) {
32  → → fprintf(stderr, "파일 읽기 오류");
33  → → exit(1);
34  → }
35
36  → // 파일로부터 정수 자료를 하나씩 읽어 연결 리스트에 추가
37  → while (!feof(fp)) {
38  → → fscanf(fp, "%d", &tmp);
39  → → list = insert_first(list, tmp);
40  → }
41
42  → // 오름차순으로 버블 정렬 후, 연결 리스트 출력
43  → bubble_sort(list);
44  → print_list(list);
45
46  → clear(list); → // 연결 리스트의 모든 노드 해제
47  → fclose(fp); → // 파일 닫기
48  }
49
50  // 리스트의 첫번째 노드로 삽입
51  ListNode *insert_first(ListNode *head, element value) {
52  → ListNode *p = (ListNode *) malloc(sizeof(ListNode)); // ListNode 노드 구조체 동적 할당
53
54  → p->data = value; → // 데이터 삽입
55  → p->link = head; → // 노드의 link에 head 포인터의 주소 대입
56  → head = p; → → // head 포인터를 새로운 노드 p의 주소 대입
57  → return head; → → // 변경된 head 포인터 반환
58  }

```

```

60
61 void print_list(ListNode* head) {
62     ListNode* p;
63     for (p = head; p->link != NULL; p = p->link) { // head부터 마지막 노드의 전 노드까지 순회하여 출력
64         printf("%d->>", p->data);
65         printf("%d\n", p->data); // 마지막 노드의 데이터는 줄바꿈과 함께 출력
66     }
67
68     // 버블 정렬을 이용한 오름차순 정렬
69 void bubble_sort(ListNode* head) {
70     ListNode* p = head, *q = NULL;
71     element tmp;
72     int sorted;
73
74     if (head == NULL) return; // 빈 리스트의 경우
75     do {
76         p = head;
77         sorted = 0; // 정렬 여부 확인 (초깃값=0)
78         while (p->link != q) { // p의 다음 노드가 q일 때까지 반복하여 현재 노드와 다음 노드의 데이터 크기 비교
79             if (p->data > p->link->data) { // 다음 노드의 데이터가 현재 노드의 데이터보다 큰 경우 두 값을 변경
80                 tmp = p->data;
81                 p->data = p->link->data;
82                 p->link->data = tmp;
83                 sorted = 1; // 한 번이라도 정렬이 된 경우, 값을 1로 변경
84             }
85             p = p->link; // 다음 노드로 이동
86         }
87         q = p; // 순회를 마무리한 마무리 p 포인터 값을 q에 대입
88     } while (sorted); // sorted가 1일 때까지 반복
89 }
90
91 // 첫 노드부터 차례대로 리스트의 모든 노드 메모리 해제 (delete_first)
92 void clear(ListNode* head) {
93     ListNode* removed = head; // 메모리를 해제하고자 하는 노드를 가리키는 포인터
94
95     if (head == NULL) return; // 빈 리스트의 경우
96     while (head != NULL) { // head 포인터부터 순회하며 차례대로 노드의 메모리 해제
97         removed = head; // 제거하고자 하는 노드의 주소는 첫 head 포인터
98         head = removed->link; // head 포인터에 제거하는 노드의 다음 노드의 주소 대입
99         free(removed); // removed 포인터가 가리키는 노드 메모리 해제
100     }
101 }

```

1.3 소스 코드 분석

```

/*
→  작성자: .이예빈 (20204059)
→  작성일: .2021.09.13
→  프로그램명: .파일에서 정수 자료를 읽어, 동적으로 메모리를 할당 받아 연결 리스트에 저장하고,
→  →  →  리스트에 저장된 정수들을 버블 정렬을 사용하여 오름차순으로 정렬하여 출력한다.
*/

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

// 연결 리스트의 노드 구조체 선언
typedef int element;
typedef struct ListNode {
→  element data;
→  struct ListNode* link;
} ListNode;

```

1. 작성자, 작성일, 프로그램명을 주석에 작성한다. 이후, 표준 입출력 헤더 파일과 메모리 동적 할당에 필요한 malloc() 함수를 위한 표준 라이브러리 헤더 파일을 추가하도록 한다. 그리고 연결 리스트의 노드에 대한 구조체를 선언한다. 정수형 타입인 element 를 사용자 정의 자료형으로 선언하였고, ListNode 를 element 형의 data, 그리고 다음 노드를 가리킬 포인터인 link 를 멤버를 가지는 구조체로 선언하였다.

```

// 함수 원형 선언
ListNode* insert_first(ListNode* head, element value);
void print_list(ListNode* head);
void bubble_sort(ListNode* head);
void clear(ListNode* head);

int main() {
→  FILE* fp; →  →  →  →  // 파일 포인터
→  ListNode* list = NULL; →  // 연결 리스트의 헤드 포인터
→  element tmp; →  →  →  // 데이터를 임시로 저장할 element 형 변수 tmp
}

```

2. 프로그램에서 사용하게 되는 함수 원형을 작성하였다. 정수 요소를 리스트의 첫 노드로 삽입하는 insert_first, 리스트 전체를 출력하는 print_list, 리스트를 버블 정렬을 이용하여 오름차순으로 정렬하는 bubble_sort, 그리고 리스트의 모든 노드를 차례대로 해제하는 clear 함수가 사용된다. 그리고 main 함수가 시작된다. 파일 포인터 fp 를 선언하였고, 연결 리스트의 헤드 포인터인 list, 그리고 파일로부터 읽을 정수 자료를 임시로 저장하는 element 형의 변수 tmp 를 선언하였다.

```

→ //파일 열기 (오류 코드 포함)
→ fp=fopen("data.txt", "rt");
→ if (fp==NULL) {
→     fprintf(stderr, "파일 읽기 오류");
→     exit(1);
→ }

→ //파일로부터 정수 자료를 하나씩 읽어 연결 리스트에 추가
→ while (!feof(fp)) {
→     fscanf(fp, "%d", &tmp);
→     list=insert_first(list, tmp);
→ }

```

3. 정수 자료들이 저장되어 있는 data.txt 파일을 읽기 모드로 연다. 파일 읽기를 실패한 경우에 대한 코드도 추가하였다. 파일 열기가 성공하면, 파일로부터 정수 자료를 하나씩 읽게 된다. 파일 포인터가 파일의 끝을 가리킬 때까지, fscanf 함수가 정수 변수를 하나씩 읽어 tmp 에 저장하고, 이후 리스트의 헤드 포인터와 tmp 에 저장된 정수가 insert_first 함수의 매개 변수로 전달되고, 변경된 헤드 포인터가 반환되어 다시 list 에 저장된다. (insert_first 함수의 동작 방식은 아래에 자세히 작성되어 있다.)

```

→ //오름차순으로 버블 정렬 후, 연결 리스트 출력
→ bubble_sort(list);
→ print_list(list);

→ clear(list); → //연결 리스트의 모든 노드 해제
→ fclose(fp); → //파일 닫기
}

```

4. list 리스트를 오름 차순 버블 정렬하는 bubble_sort, 그리고 정렬된 리스트를 출력하는 print_list 가 차례대로 호출된다. 이후 리스트의 모든 노드에 대한 메모리를 해제하는 clear 함수가 호출되고, 마지막으로 파일을 닫는 fclose 함수가 호출되며 프로그램은 종료된다.

```

//리스트의 첫번째 노드로 삽입
ListNode* insert_first(ListNode* head, element.value) {
→     ListNode* p=(ListNode*)malloc(sizeof(ListNode)); → //ListNode 노드 구조체 동적 할당

→     p->data=element.value; → //데이터 삽입
→     p->link=head; → //노드의 link에 head 포인터의 주소 대입
→     head=p; → //head 포인터를 새로운 노드 p의 주소 대입
→     return head; → //변경된 head 포인터 반환
}

```

5. 연결 리스트의 헤드 포인터와 삽입하고자 하는 element(정수)형 자료를 전달받아, 새로운 노드에 대한 메모리를 할당하여 리스트의 첫 부분에 삽입하는 insert_first 함수이다. ListNode 노드 크기만큼의 메모리를 malloc 함수를 이용하여 할당 받아 해당 메모리 주소를 p 포인터에 대입한다. 해당 노드의 data 멤버에는 인수로 전달받은 정수 자료를 저장하고, link 멤버에는 기존 head 의 주소를 대입한다. 이후 head 포인터에는 현재 p 가 가리키는, 새로운 노드의 주소를

대입하여 연결 리스트의 첫 노드가 되도록 한다. 마지막으로, 변경된 head 포인터를 반환한다. 이로써, 새로 생성되는 노드는 리스트의 첫 노드로서 삽입이 되는 것이다.

```
void print_list(ListNode* head) {
    ListNode* p;
    for (p = head; p->link != NULL; p = p->link) { // head부터 마지막 노드의 전 노드까지 순회하여 출력
        printf("%d-> ", p->data);
        printf("%d\n", p->data); // 마지막 노드의 데이터는 줄바꿈과 함께 출력
    }
}
```

6. 연결 리스트를 순회하며 모든 노드의 정수 데이터를 출력하는 print_list 함수이다. 형식에 맞게 출력해야 하므로, 가장 마지막 노드의 이전 노드까지 순회하도록 for 문이 돌며 정수를 출력하고, 가장 마지막 노드의 정수 값은 화살표가 아닌 줄바꿈 문자와 함께 출력 된다.

7. 다음으로는 오름차순으로 정렬을 하는 버블 정렬 bubble_sort 함수이다.

```
// 버블 정렬을 이용한 오름차순 정렬
void bubble_sort(ListNode* head) {
    ListNode* p = head, *q = NULL;
    element tmp;
    int sorted;
```

7-1. 먼저 노드를 가리키는 포인터 p 와 q 를 선언하였다. p 는 head 포인터가 가리키는 노드부터 시작하여, 각 노드를 순회하며 다음 노드의 데이터와 현재 노드의 데이터를 비교하며 정렬을 하는데 사용되는 포인터이다. 포인터 q 는 null 포인터부터 시작하여, 가장 끝 노드를 가리키는 노드로서, 순회하는 포인터 p 가 어디까지 순회해야 할지 그 범위를 결정하는데 사용되기도 하는 포인터이다. 반복하며 정렬을 거듭할 때마다 포인터 q 는 가장 마지막에 오게 되는 노드, 즉 가장 큰 정수 값을 가지는 마지막 노드를 가리키게 된다. 현재 노드의 정수와 다음 노드의 정수 값을 교환하기 위해, 임시로 값을 담아 저장해 놓을 변수 tmp 를 선언하였다. 마지막으로, 한 번 p 가 순회할 때마다 두 노드의 정수가 교환되는 경우의 플래그 값을 확인하도록 하는 정수형 변수 sorted 를 선언하였다. sorted 는 p 가 각 노드를 순회하기 시작 전마다 0 으로 초기화되었다가 두 값의 교환이 발생하는 경우, 1 로 그 값이 변경된다.

아래 코드에 버블 정렬의 동작 원리가 나타나 있다.

```

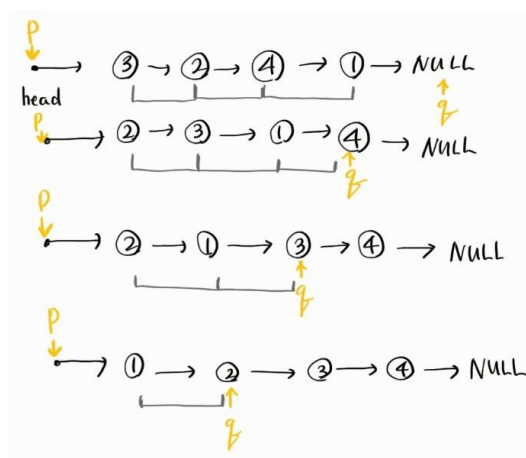
→ if (head == NULL) .return; // 빈 리스트의 경우
→ do {
→   p = head; → → → → → → // head 포인터에 대하여 정렬 반복
→   sorted = 0; → → → → → // 정렬 여부 확인 (초기값=0)
→   while (p->link != q) { → → → // p의 다음 노드가 q일 때까지 반복하여 현재 노드와 다음 노드의 데이터 크기 비교
→     if (p->data > p->link->data) { // 다음 노드의 데이터가 현재 노드의 데이터보다 큰 경우 두 값을 변경
→       tmp = p->data;
→       p->data = p->link->data;
→       p->link->data = tmp;
→       sorted = 1; → → → // 한 번이라도 정렬이 된 경우, 값을 1로 변경
→     }
→     p = p->link; → → → // 다음 노드로 이동
→   }
→   q = p; → → → // 순회를 마무리한 후 p 포인터 값을 q에 대입
→ } while (sorted); → → → // sorted가 1일 때까지 반복
}

```

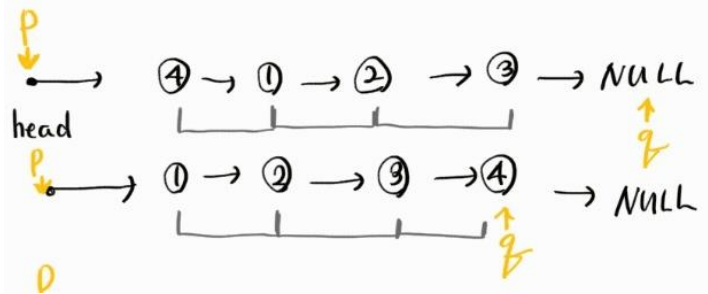
7-2. 빈 리스트가 아닌 경우, p는 head 포인터가 가리키는 첫번째 노드부터 다음 노드가 q인 노드일 때까지 순회하며, 현재 노드의 정수 값과 다음 노드의 정수 값을 비교하여 현재 값이 더 큰 경우 다음 값과 교환하기를 반복한다. 이러한 과정은 한 번 순회할 때마다, 더 이상 값이 바뀌는 상황이 일어나지 않는 경우가 생길 때까지 반복된다. 값이 바뀌지 않는 경우는, sorted 플래그 값을 통해 확인할 수 있다. 한 번 순회할 때마다, sorted 정수는 0의 초기값을 가지고 있다가 두 노드의 정수 값이 교환되는 경우에 그 값이 1로 변경되기 때문이다. 한 번의 순회가 끝나면, 마지막으로 도달한 노드의 정수는 현재까지 순회한 정수 값들 중 가장 큰 값이 된다.

맨 처음 q는 null의 초기값을 가지고 있으므로, p는 리스트의 마지막 노드까지 순회하게 된다. p가 첫번째로 순회할 때에는, 가장 큰 정수 값을 맨 마지막 노드로 끌고 오게 된다. 따라서 마지막 노드를 가리키는 p의 주소가 q에 대입되고, 다음 순회 때에는 두번째로 큰 정수가 끝에서 두번째 노드로 오게 된다. 이러한 과정을 반복하는데, 이는 q가 가리키는 것이 null 값을 의미하는 리스트의 마지막부터 시작하여, 차례대로 앞의 노드로 이동하게 되는 것이라 설명할 수 있다.

이러한 과정을 간단히 다음과 같이 그림으로 표현해 보았다. 매 순회 때마다, p는 head 포인터 값부터 시작하여 q노드까지 이동하며 두 노드의 정수 값을 오름차순이 되도록 교환하기를 반복한다.



그리고, 다음과 같이 두번째 순회의 과정에서 더 이상 교환이 일어나지 않는 경우, sorted 플래그 값이 0 이므로, do~while 반복문은 종료된다. sorted 플래그 값의 확인을 통해, 오름차순 정렬이 완료되어 더 이상의 순회는 반복하지 않게 되어 정렬 알고리즘의 효율성을 보다 높여 보았다.




```
//첫·노드부터·차례대로·리스트의·모든·노드·메모리·해제·(delete_first)
void clear(ListNode* head) {
    → ListNode* removed = head; → //·메모리를·해제하고자·하는·노드를·가리키는·포인터

    → if (head == NULL) return; → //·빈·리스트의·경우
    → while (head != NULL) { → → //·head·포인터부터·순회하며·차례대로·노드의·메모리·해제
    → → removed = head; → → //·제거하고자·하는·노드의·주소는·첫·head·포인터
    → → head = removed->link; → //·head·포인터에·제거하는·노드의·다음·노드의·주소·대입
    → → free(removed); → → //·removed·포인터가·가리키는·노드·메모리·해제
    → }
}
```

8. 마지막으로 리스트의 모든 노드에 할당된 메모리를 해제하는 clear 함수이다. 메모리를 해제하고자 하는 노드를 가리키는 포인터를 removed 이다. removed 는 반복문이 실행될 때마다 head 포인터, 즉 첫번째 노드의 주소로 값이 변경되어 해당 노드의 메모리를 해제한다. free 함수로 해제를 하기 이전에는, head 포인터에는 다시 다음 노드의 주소를 대입하게 된다. Head 포인터가 리스트의 마지막인 null 값을 갖게 되면, 모든 노드의 메모리를 해제하게 된 것이므로 반복문이 종료된다.


1.4 실행창

실행 결과 1

 data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)


4 5 9 7 2 1 6 8

 Microsoft Visual Studio 디버그 콘솔

1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 ->


C:\Users\이예빈\Desktop\CSE 2021-1
로되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

실행 결과 2

 data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)


8 4 6 3 1 2 5

 Microsoft Visual Studio 디버그 콘솔

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 8


C:\Users\이예빈\Desktop\CSE 2021-1\과목들
되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

실행 결과 3

 data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

245 93 24 92 73 192 335 17

 Microsoft Visual Studio 디버그 콘솔

17 -> 24 -> 73 -> 92 -> 93 -> 192 -> 245 -> 335

C:\Users\이예빈\Desktop\CSE 2021-1\과목들
되었습니다(코드: 0개).

1.5 느낀점

배열에 대한 메모리를 동적 할당했던 1 주차 실습 과제를 할 때에는 먼저 파일을 스캔하여 몇 명의 학생 데이터가 저장되어 있는지 확인한 후, 학생 수만큼의 크기의 배열을 동적으로 할당하고 다시 파일을 읽어야 한다는 약간의 번거로움이 존재했다. 그에 비해 이번 2 주차 과제에서 사용한 연결 리스트는 각 노드의 링크를 연결할 수 있다는 점에서, 파일을 여러 번 읽지 않아도 된다는 리스트의 장점을 다시 한번 확인할 수 있었다. 비록 각 노드의 링크를 연결해야 한다는 점에서 코드 작성의 실수가 빈번하게 일어나기도 하지만, 데이터의 크기가 매우 커지는 경우 파일 입출력 부분에서의 시간 복잡도가 매우 줄어들기 때문에 리스트를 활용하는 것에는 큰 장점이 존재한다.

이번 2 주차 과제에서는 버블 정렬 부분에서 어떻게 하면 조금 더 효율적으로 코드를 작성할 수 있을까 고민을 많이 하게 되었다. 교재에 나와 있는 코드, 그리고 검색하여 나오는 코드 자료들을 바탕으로, 여러 경우를 대비할 수 있는 방향으로 버블 정렬 코드를 작성해 보았다. 그저 데이터의 개수를 사용해서 이중 반복문으로 버블 정렬을 구현해 볼 수도 있겠지만, 이번에는 포인터를 이용하여 각 순회 때마다 구해지는 가장 큰 값을 가진 마지막 노드의 주소로 포인터 값을 변경하여 다음 순회의 끝을 결정할 수 있도록 하는 부분, 그리고 더 이상의 데이터 교환이 일어나지 않는 경우를 고려하는 등 여러 가지 경우를 생각하여 버블 정렬을 구현해 보았다.

다른 객체 지향 프로그래밍 언어를 배우며 예외 처리를 하는 부분도 고려하게 되는데, 이번 자료구조 시간에서 프로그램을 작성할 때에도 조금 더 다양한 경우를 생각하여 코드를 작성하게 되는 것 같다. 물론 코드가 단순함을 조금 벗어나게 되는 경우도 생기지만, 아직은 배우는 과정이기 때문에 효율성을 벗어나지 않는 영역에서 다양한 시도를 해보고 싶다.

버블 정렬에 대해 깊이 고민하다 보니, 앞으로 2 학기 자료구조 시간에 배우게 될 다양한 정렬 알고리즘이 기대가 된다.

2. 파일 입출력, 동적 할당과 단순 연결리스트, 정렬을 이용한 학생 성적 관리 프로그램

■ 성적 관리 프로그램

- 성적 관리 프로그램을 만들려고 한다. 이때 학생의 정보를 가진 구조체로 이루어진 연결리스트를 이용하고 학생 자료는 data.txt에 저장되어있다. 또한 추가한 학생의 순서대로 연결리스트에 정보를 저장한다. 성적 관리 프로그램에 들어가는 기능들은 종료, 학생 추가, 검색 그리고 목록 보기가 있다. 아래의 조건까지 만족하는 프로그램을 작성하라.

- 데이터는 data.txt를 사용할 것
- 학생 구조체 정의는 Student로 정의
- 초기화와 메모리 해제는 반드시 할 것
- 예시와 같은 표 양식으로 출력
- 이름은 4글자까지 가능함
- 동적 할당 이용
- 검색 시 학생의 점수 총합 및 평균 출력

data.txt

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
 20170245 김지연 40 40 100
 20170285 김준섭 60 20 85
 20170488 전희원 80 40 20

2.1 문제 분석

이 문제는 파일에 저장되어 있는 학생 데이터를 연결 리스트에 저장하고, 또 사용자로부터 입력 받는 메뉴에 따라 학생을 추가하고, 학번에 따라 학생을 검색하고, 전체 리스트의 목록을 볼 수 있도록 하는 성적 관리 프로그램이다. 다음은 구현을 위해 고려해야 할 부분이다.

1. 구조체

- 학생의 정보를 담는 구조체와 리스트의 노드를 구현하기 위한 구조체가 필요하다.
- 학생 구조체는 학번, 이름, 세 과목의 점수, 그리고 총합과 평균으로 이루어져 있다. 이 문제 같은 경우, 검색시에만 총합과 평균을 필요로 한다. 그러나 '학생 성적 관리 프로그램'이라는 점을 고려하여 총합과 평균에 대한 값도 구조체의 멤버로 포함하였다. 이름이 4 글자까지 가능하다는 점을 고려하여, 학생의 이름에 대한 구조체 멤버를 최대 9 비트 길이의 문자배열로 설정해야 한다.
- 리스트의 노드 구조체는 학생 구조체에 대한 데이터 필드, 그리고 다음 노드를 가리키는 자체 참조 구조체 포인터인 링크 필드로 구성된다.

2. 파일 입출력

- 파일에 데이터가 입력되어 있는 형식에 따라, 파일을 읽을 때마다 학생 한 명의 데이터에 대하여, 학생 구조체의 임시 변수의 각 멤버에 학생의 학번, 이름, 그리고 총합과 평균을 구한 값을 저장하도록 한다. 이후 이 구조체 변수에 저장된 한 학생의 데이터는, 연결 리스트의 헤드 포인터의 주소, 그리고 리스트의 마지막 노드를 가리키는 포인터의 주소와

함께 전달되어 새로운 학생 노드에 대하여 동적 할당을 하고 리스트의 끝에 노드를 삽입하는 함수 내에서 사용되도록 한다.

3. 동적 할당과 연결 리스트, 이중 포인터를 통해 데이터를 리스트의 끝에 삽입하는 함수 구현

- 학생 노드가 리스트의 끝에 삽입되는 프로그램이므로, 리스트 전체를 순회하며 마지막에 삽입하는 방법이 아닌, 가장 마지막 노드에 대한 포인터를 이용하는 방법이 시간 복잡도 부분에서 매우 효율적이다. 따라서 파일로부터 데이터를 읽거나 사용자로부터 직접 입력 받은 후 리스트에 추가할 때, 마지막 노드를 가리키는 포인터의 주소(이중 포인터)를 새로운 학생 노드를 동적 할당하여 리스트의 끝에 삽입하는 함수의 인자로 전달하는 것을 고려해야 한다.
- head 포인터가 NULL 값일 때와 그렇지 않을 때를 구분하여 적절히 조건문을 활용해야 한다. 빈 리스트인 경우에는, head 포인터에 첫번째 새로운 노드의 주소를 대입한다. 빈 리스트가 아닌 경우에는, 마지막 노드를 가리키는 포인터를 통해 이전의 마지막 노드의 링크 필드에 새로운 노드 주소를 대입한다. 마지막으로는 항상, 마지막 노드를 가리키는 포인터에 새로운 노드의 주소를 대입하여, 다음번의 노드 추가 시에 이 포인터를 활용할 수 있도록 한다.

4. 학생 데이터를 사용자로부터 직접 입력 받는 경우

- 학생 데이터는 파일로부터 읽거나 사용자로부터 직접 입력 받는 방법으로 얻을 수 있게 된다. 파일로부터 읽는 경우 학생 구조체의 임시 변수에 데이터를 저장하게 된다.
- 마찬가지로, 사용자로부터 직접 데이터를 입력 받는 경우에도 구조체의 임시 변수에 데이터를 저장한다. 사용자로부터 데이터를 입력 받는 부분을 함수로 따로 구현하여, 입력 받은 학생 데이터에 대한 구조체를 반환 받아 이를 임시 변수에 저장하도록 한 후, 리스트에 삽입하는 함수의 인자 중 하나로 전달할 수 있도록 하였다.

5. 리스트에서 값 탐색

- 학생의 학번을 검색하는 함수는, 리스트의 헤드 포인터가 가리키는 첫번째 노드부터 순회하여 리스트에 저장되어 있는 학생 중 입력 받은 학번의 학생이 있는 경우 해당 학생의 데이터를 출력하도록 구현하였다. 끝까지 순회한 후 탐색에 실패한 경우에 검색 실패했다는 것을 출력하도록 한다.

6. 리스트 출력, 리스트 메모리 해제

- 리스트에서 각 학생 데이터를 출력하는 것과 리스트의 모든 노드에 대한 메모리를 해제하는 것은 리스트의 첫 노드부터 끝 노드까지 순회한다는 부분에서 동일하다. 리스트를 출력하고, 리스트의 메모리를 해제하는 부분도 따로 함수로 구현하였다.

2.2 소스 코드

```

1  /*
2  → 작성자: .이예빈 (20204059)
3  → 작성일: .2021.09.13
4  → 프로그램명: .파일로부터 .학생의 .정보들을 .읽어 .구조체와 .동적 .할당을 .이용해 .연결 .리스트에 .저장하고,
5  → → → 사용자로부터 .입력 .받는 .메뉴에 .따라 .작동하는 .학생 .성적 .관리 .프로그램
6  */
7
8  #include <stdio.h>
9  #include <string.h>
10 #include <stdlib.h>
11
12 // .학생 .구조체
13 typedef struct Student {
14 → int id;
15 → char name[9]; // .이름은 .4글자까지 .가능
16 → int math, kor, eng;
17 → int sum;
18 → double avg;
19 } Student;
20
21 // .연결 .리스트 .노드 .구조체
22 typedef struct ListNode {
23 → Student data;
24 → struct ListNode *link;
25 } ListNode;
26
27 // .함수 .원형
28 void print_menu();
29 void insert_last(ListNode** head, ListNode** prev, Student new_data);
30 void print_list(ListNode* head);
31 Student add_data();
32 void search(ListNode* head);
33 void clear(ListNode* head);
34
35 int main() {
36 → FILE* fp; → → → // .파일 .포인터
37 → ListNode* head = NULL, *last = NULL; → // .학생 .정보 .연결 .리스트의 .head 포인터
38 → Student tmp; → → // .파일로부터 .읽는 .학생 .데이터 .저장하는 .임시 .구조체
39 → int getMenu; → → // .메뉴-입력 .받는 .정수
40
41 → // .파일 .읽기 .모드로 .읽기
42 → fp = fopen("data.txt", "rt");
43 → if (fp == NULL) {
44 → → fprintf(stderr, ".파일 .열기 .실패\n");
45 → → exit(1);
46 → }
47
48 → // .파일에서 .읽은 .학생 .데이터 .리스트의 .마지막에 .삽입하기
49 → while (!feof(fp)) {
50 → → fscanf(fp, "%d%s%d%d", &tmp.id, tmp.name, &tmp.math, &tmp.eng, &tmp.kor);
51 → → tmp.sum = tmp.math + tmp.eng + tmp.kor;
52 → → tmp.avg = tmp.sum / 3.0;
53 → → insert_last(&head, &last, tmp);
54 → }
55 → printf("<<성적 .관리 .프로그램>>");
56
57 → // .0번 .메뉴 .입력받을 .때까지 .반복하여 .성적 .관리 .프로그램 .실행
58 → while (1) {
59 → → print_menu();
60 → → scanf("%d", &getMenu);
61 → → if (getMenu == 0) break;

```



```

62  → switch (getMenu) {
63  → case 1: search(head); break;
64  → case 2: print_list(head); break;
65  → case 3:
66  →     tmp = add_data(); // 학생 데이터를 전달 받아 tmp에 저장
67  →     insert_last(&head, &last, tmp); // head 이중포인터, 마지막 노드 last 이중포인터
68  →     break;
69  → default: break;
70  → }
71  → }
72  → clear(head); // 리스트의 모든 노드 해제
73  → fclose(fp); // 파일 닫기
74  → }
75
76  void print_menu() {
77  → printf("\n>>-0-종료\n");
78  → printf(">>-1-검색\n");
79  → printf(">>-2-목록\n");
80  → printf(">>-3-추가\n\n");
81  → }
82
83  // 리스트의 끝에 새로운 학생 데이터 노드를 추가
84  void insert_last(ListNode** head, ListNode** prev, Student new_data) {
85  → // 새로운 노드 동적 할당 후 학생 데이터 저장
86  → ListNode* new_node = (ListNode*) malloc(sizeof(ListNode));
87  → new_node->data = new_data;
88  → new_node->link = NULL;
89
90  → if (*head == NULL) → // 빈 리스트인 경우
91  →     *head = new_node; → // 첫 헤드 포인터에 새로운 노드의 주소 저장
92  → else → // 빈 리스트가 아닌 경우
93  →     (*prev)->link = new_node; → // 가장 마지막 노드의 link에 현재 노드의 주소 저장
94
95  → *prev = new_node; // 추가된 새로운 노드 주소를 마지막 노드의 포인터를 가리키는 prev 이중 포인터에 저장
96  → }
97
98  // 리스트에 삽입하고자 하는 학생 데이터 입력받기
99  Student add_data() {
100  → Student tmp; → // 입력받는 정보들을 저장할 임시 구조체
101  → char name[9]; → // 학생의 이름을 임시로 저장할 문자 배열
102
103  → // 사용자에게 학생 정보 입력받아 tmp 구조체에 저장
104  → printf("추가할 학생 이름: ");
105  → scanf("%s", name);
106  → strcpy(tmp.name, name); // 구조체의 name 멤버에 문자열 복사
107
108  → printf("학번: "); scanf("%d", &tmp.id);
109  → printf("수학 점수: "); scanf("%d", &tmp.math);
110  → printf("영어 점수: "); scanf("%d", &tmp.eng);
111  → printf("국어 점수: "); scanf("%d", &tmp.kor);
112
113  → tmp.sum = tmp.math + tmp.eng + tmp.kor;
114  → tmp.avg = tmp.sum / 3.0;
115
116  → return tmp;
117  → }
118
119  void print_list(ListNode* head) {
120  → ListNode* p = head;
121
122  → printf("|=====|\n");

```

```

123 → printf("|·학·..·번·|·..이·름·|·수학·|·영어·|·국어·|\n");
124 → printf("|\n");
125
126 → while(p!=NULL){
127 → → printf("|·%d|·%s|·%3d·|·%3d·|·%3d·|\n", p->data.id, p->data.name,
128 → → p->data.math, p->data.eng, p->data.kor);
129 → → p=p->link;
130 → }
131 → printf("|\n");
132 }
133
134 //·학번·검색하여·학생·정보·출력
135 void search(ListNode* head){
136 → ListNode* p:=head;
137 → int id;
138
139 → printf("검색할·학번·:");
140 → scanf("%d", &id);
141
142 → while(p!=NULL){
143 → → if(p->data.id==id){ //·입력받은·id와·동일한·노드의·id가·존재한다면·출력
144 → → → printf("<·이름·:·%s·>\n", p->data.name);
145 → → → printf("<·수학·:·%3d점·>\n", p->data.math);
146 → → → printf("<·영어·:·%3d점·>\n", p->data.eng);
147 → → → printf("<·국어·:·%3d점·>\n", p->data.kor);
148 → → → printf("<·총점·:·%3d점·>\n", p->data.sum);
149 → → → printf("<·평균·:·%3d점·>\n\n", (int)p->data.avg);
150 → → → return;
151 → → }
152 → → p=p->link;
153 → }
154 → printf("학번·%d·검색·실패\n", id);
155 }
156
157 //·리스트의·모든·노드·메모리·해제
158 void clear(ListNode* head){
159 → ListNode* p, *next; → //·메모리를·해제할·노드를·가리키는·포인터·p와·다음·노드를·가리키는·포인터·next
160
161 → p:=head; → → → //·p가·head부터·null까지·노드를·순회하며·메모리·해제
162 → while(p!=NULL){ → → //·p가·null·이·아닐·때까지·반복
163 → → next:=p->link; → → //·다음·노드·주소를·next에·저장
164 → → free(p); → → //·p가·가리키는·노드의·메모리·해제
165 → → p=next; → → //·삭제될·다음·노드·주소를·p의·값으로·변경
166 → }
167 }

```

2.3 소스 코드 분석

```

/*
→ 작성자: ·이예빈 (20204059)
→ 작성일: ·2021.09.13
→ 프로그램명: ·파일로부터 ·학생의 ·정보들을 ·읽어 ·구조체와 ·동적 ·할당을 ·이용해 ·연결 ·리스트에 ·저장하고,
→   →   →   사용자로부터 ·입력 ·받는 ·메뉴에 ·따라 ·작동하는 ·학생 ·성적 ·관리 ·프로그램
*/

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

```

1. 주석에 작성자, 작성일, 프로그램명을 작성한다. 표준 입출력, 동적 할당하는 malloc()을 위한 표준 라이브러리, 문자열을 복사하는 strcpy()를 위한 string 헤더 파일을 추가한다.

```

// ·학생 ·구조체
typedef struct Student {
→   int ·id;
→   char ·name[9]; ·// ·이름은 ·4글자까지 ·가능
→   int ·math, ·kor, ·eng;
→   int ·sum;
→   double ·avg;
} Student;

// ·연결 ·리스트 ·노드 ·구조체
typedef struct ListNode {
→   Student ·data;
→   struct ·ListNode* ·link;
} ListNode;

```

2. 학번, 4 글자까지의 이름을 저장 가능한 길이 9 비트의 문자배열, 수학/국어/영어 점수 그리고 총점을 저장하는 정수형, 마지막으로 평균값을 저장하는 실수형을 구조체의 멤버로 포함하여 Student 형 구조체를 선언한다. 그리고 이러한 구조체를 데이터로, 그리고 다음 노드를 가리킬 포인터를 멤버로 포함하는 노드를 ListNode 형 구조체로 선언한다.

```

// ·함수 ·원형
void ·print_menu();
void ·insert_last(ListNode** ·head, ·ListNode** ·prev, ·Student ·new_data);
void ·print_list(ListNode* ·head);

```

```

Student·add_data();
void·search(ListNode*·head);
void·clear(ListNode*·head);

int·main()·{
→   FILE*·fp; → → → → //·파일·포인터
→   ListNode*·head·=·NULL, *·last·=·NULL; → //·학생·정보·연결·리스트의·head·포인터
→   Student·tmp; → → → //·파일로부터·읽는·학생·데이터·저장하는·임시·구조체
→   int·getMenu; → → → //·메뉴·입력·받는·정수
}

```

4. 프로그램에서 사용되는 함수들의 원형을 미리 main 함수에 앞서 선언하였다. 메뉴들을 출력하는 print_menu, 새로운 노드 구조체(학생 구조체)의 메모리를 동적 할당하고 리스트의 마지막에 이를 삽입하는 insert_last, 리스트 전체를 출력하는 print_list, 새로운 노드의 데이터(학생 정보)를 사용자로부터 입력받는 add_data, 학번을 검색하여 리스트 목록의 학생 데이터를 출력하는 search, 그리고 리스트의 모든 노드의 메모리를 해제하는 clear 함수의 원형이 나타나 있다.

원형 선언 후에는 main 함수가 시작된다. 파일 포인터, 학생 정보 연결 리스트를 가리키는 head 포인터와 연결 리스트의 마지막 노드를 가리키는 last 포인터, 그리고 파일에 저장되어 있는 학생 데이터를 임시로 저장할 Student 형 구조체 tmp, 마지막으로 사용자로부터 메뉴를 입력 받을 정수형 getMenu 를 선언하였다.

```

→ //·파일·읽기·모드로·열기
→ fp·=·fopen("data.txt", "rt");
→ if (fp·==·NULL)·{
→   fprintf(stderr, "파일 열기 실패\n");
→   exit(1);
→ }

→ //·파일에서·읽은·학생·데이터·리스트의·마지막에·삽입하기
→ while (!feof(fp))·{
→   fscanf(fp, "%d·%s·%d·%d", &tmp.id, &tmp.name, &tmp.math, &tmp.eng, &tmp.kor);
→   tmp.sum·=·tmp.math·+·tmp.eng·+·tmp.kor;
→   tmp.avg·=·tmp.sum·/·3.0;
→   insert_last(&head, &last, tmp);
→ }

```

5. 파일을 읽기 모드로 열고, 열기를 실패한 경우의 코드를 작성한다. 파일을 처음부터 끝까지 읽으며, Student 형 구조체 tmp 임시 변수에 파일에 저장된 학생 데이터(학번 id, 이름, 수학/영어/국어 성적)를 차례대로 구조체의 각 멤버에 저장한다. 입력된 세 과목의 성적들을 더하여 sum 멤버 변수에 저장하고, 평균 값을 구하여 avg 에도 저장한다. 완성된 하나의 학생 구조체를 insert_last 에 리스트의 head 포인터 주소, 그리고 last 포인터의 주소와 함께 전달하여 리스트의 끝에 새로운 학생 노드를 추가하도록 한다.

```

→ printf("<<성적.관리.프로그램>>");
→ //0번.메뉴.입력받을.때까지.반복하여.성적.관리.프로그램.실행
→ while(1){
→     print_menu();
→     scanf("%d",&getMenu);
→     if(getMenu==0) break;
→     switch(getMenu){
→     case 1: search(head); break;
→     case 2: print_list(head); break;
→     case 3:
→         tmp=add_data(); //학생.데이터를.전달.받아.tmp에.저장
→         insert_last(&head,&last,tmp); //head.이중포인터,.마지막.노드.last.이중포인터
→         break;
→     default: break;
→     }
→ }

```

6. 성적 관리 프로그램의 메뉴를 출력하고, 사용자로부터 입력 받는 getMenu 값에 따라 각 함수들이 동작을 하게 된다. 0의 값이면 반복문이 종료되고, 1의 값이면 리스트에서 학생 검색을 하는 search가 호출되고, 2의 값이면 리스트를 출력하는 print_list가 호출되고, 3의 값이면 사용자로부터 새로운 학생 데이터를 입력 받아 학생 구조체를 반환하는 add_data 함수의 반환 값을 tmp에 저장하고 이를 리스트의 끝에 추가하는 insert_last를 호출하게 된다.

```

→ clear(head); //리스트의.모든.노드.해제
→ fclose(fp); //파일.닫기
}

```

7. 해당 반복문이 종료된다면, clear 함수를 통해 리스트의 모든 노드를 해제하고, 마지막으로 파일을 닫아 모든 프로그램의 실행을 종료하도록 한다.

다음으로는 각 함수들의 코드를 살펴보자.

```

void print_menu(){
→     printf("\n>>-0-종료\n");
→     printf(">>-1-검색\n");
→     printf(">>-2-목록\n");
→     printf(">>-3-추가\n\n");
}

```

8. 메뉴들이 출력되는 함수이다. 0: 종료, 1: 검색, 2: 목록, 3: 추가. main 함수 내의 반복문이 실행될 때마다 메뉴판이 출력된다.

```

//.리스트의.끝에.새로운.학생.데이터.노드를.추가
void insert_last(ListNode**head, ListNode**prev, Student new_data) {
→ //새로운.노드.동적.할당.후.학생.데이터.저장
→ ListNode*new_node=(ListNode*)malloc(sizeof(ListNode));
→ new_node->data=new_data;
→ new_node->link=NULL;

→ if(*head==NULL) → → → → //빈.리스트인.경우
→ → *head=new_node; → → → //첫.헤드.포인터에.새로운.노드의.주소.저장
→ else → → → → → //빈.리스트가.아닌.경우
→ → (*prev)->link=new_node; → //가장.마지막.노드의.link에.현재.노드의.주소.저장

→ *prev=new_node; //추가된.새로운.노드.주소를.마지막.노드의.포인터를.가리키는.prev.이중.포인터에.저장
}

```

9. head 포인터의 주소와 이전에 가장 마지막 노드의 주소를 저장했던 prev 포인터의 주소, 그리고 새로운 학생 구조체를 매개 변수로 전달받아 노드의 메모리를 동적 할당하고 리스트의 끝에 노드를 추가하는 insert_last 함수이다. head 와 prev 를 이중 포인터 값으로 전달 받게 되어, insert_last 를 호출한 main 함수에서도 insert_last 함수 내에서 변경되는 포인터의 값들을 기억할 수 있게 된다.

먼저 새로 추가할 학생 노드에 대한 메모리를 동적 할당하여 new_node 포인터에 해당 노드의 주소를 대입한다. new_node 의 데이터 필드 data 에 세번째 매개 변수로 전달받은 학생 구조체 new_data 가, 그리고 다음 노드를 가리키는 link 필드에는 NULL 값이 저장된다.

만약 *head 가 NULL 값으로 빈 리스트인 경우라면, *head 에는 새로 동적 할당한 new_node 의 주소를 대입한다. 그렇지 않은 경우, 이전의 마지막 노드의 주소를 담는 *prev 의 링크 필드에 동적 할당한 new_node 의 주소를 대입하여 이전의 마지막 노드와 새로 동적 할당한 노드가 연결되도록 한다.

마지막으로는, 새로 동적 할당한 new_node 가 끝 노드가 되므로 *prev 값에 new_node 의 주소를 대입한다.

```

//리스트에 삽입하고자 하는 학생 데이터 입력받기
Student add_data() {
    Student tmp; //입력받는 정보들을 저장할 임시 구조체
    char name[9]; //학생의 이름을 임시로 저장할 문자배열

    //사용자에게 학생 정보 입력받아 tmp 구조체에 저장
    printf("추가할 학생 이름: ");
    scanf("%s", name);
    strcpy(tmp.name, name); //구조체의 name 멤버에 문자열 복사

    printf("학번: "); scanf("%d", &tmp.id);
    printf("수학 점수: "); scanf("%d", &tmp.math);
    printf("영어 점수: "); scanf("%d", &tmp.eng);
    printf("국어 점수: "); scanf("%d", &tmp.kor);

    tmp.sum = tmp.math + tmp.eng + tmp.kor;
    tmp.avg = tmp.sum / 3.0;

    return tmp;
}

```

10. 사용자로부터 직접 학생 데이터를 입력 받아 새로운 학생 구조체를 반환하는 add_data 함수이다. 파일로부터 데이터를 입력 받을 때와 마찬가지로, 선언한 임시 Student 구조체 tmp 의 각 멤버에 이름, 학번, 과목의 점수들, 그리고 계산한 총합과 평균값을 저장하도록 한다. 이름과 같은 경우에는, 사용자로부터 입력 받는 문자열을 우선적으로 char 배열인 name 에 저장하고, 이를 구조체의 name 멤버 문자열에 복사하여 저장하도록 한다. 마지막으로, 완성된 tmp 학생 구조체가 반환된다.

```

void print_list(ListNode* head) {
    ListNode* p = head;

    printf("=====|\n");
    printf("| 학번 | 이름 | 수학 | 영어 | 국어 |\n");
    printf("=====|\n");

    while (p != NULL) {
        printf("| %d | %s | %3d | %3d | %3d |\n", p->data.id, p->data.name,
            p->data.math, p->data.eng, p->data.kor);
        p = p->link;
    }
    printf("=====|\n\n");
}

```

11. 포인터가 리스트의 head 부터 마지막 null 까지 순회하며 리스트 전체 목록의 학생 정보들을 출력하는 함수이다. 형식에 맞게 적절히 공백을 활용하여, 각 노드의 data 필드의 멤버들을 출력하도록 한다.

```

//·학번·검색하여·학생·정보·출력
void search(ListNode* head)·{
    →  ListNode* p·=·head;
    →  int id;

    →  printf("검색할·학번·:·");
    →  scanf("%d",·&id);

    →  while(p·!=·NULL)·{
    →  →  if(p->data.id·==·id)·{·//·입력받은·id와·동일한·노드의·id가·존재한다면·출력
    →  →  →  printf("<·이름·:·%s·>\n",·p->data.name);
    →  →  →  printf("<·수학·:·%3d점·>\n",·p->data.math);
    →  →  →  printf("<·영어·:·%3d점·>\n",·p->data.eng);
    →  →  →  printf("<·국어·:·%3d점·>\n",·p->data.kor);
    →  →  →  printf("<·총점·:·%3d점·>\n",·p->data.sum);
    →  →  →  printf("<·평균·:·%3d점·>\n\n",·(int)p->data.avg);
    →  →  →  return;
    →  →  }
    →  →  p·=·p->link;
    →  →  }
    →  printf("학번·%d·검색·실패\n\n",·id);
}

```

12. 리스트 내에서 검색하고자 하는 학생의 학번을 탐색하여, 탐색에 성공한 경우 해당 학생의 정보를 출력하는 search 함수이다. 먼저 검색할 학번을 정수형으로 사용자로부터 입력 받고, ListNode 형 포인터 p 가 head 부터 끝 노드까지 순회하며, 각 학생의 데이터 필드 중 id 값을 입력 받은 id 값과 비교하여 일치하는 경우 학생 정보를 출력하도록 한다.

이때, 과제로 주어진 예시 실행창에 총점이 정수로 출력된다는 점을 고려해야 한다. 앞서 “문제 분석” 부분에서도 설명한 것과 같이, ‘성적 관리’ 프로그램이라는 점에서 구조체의 필드에 총점과 평균을 멤버로 포함하였다. 멤버로서 평균 값은 실수형 이므로, 평균을 출력하는 경우 소수점 아래 수들을 버린 정수형으로 출력될 수 있도록 형식 지정자를 정수로 설정하였다,

해당 학번의 학생 정보를 출력한 후에는 함수를 종료하도록 하고, 만약 전체 리스트의 순회가 끝나 해당 학번의 학생을 탐색하지 못한 경우에는 검색에 실패했음을 출력하도록 한다.

```

//·리스트의·모든·노드·메모리·해제
void clear(ListNode* head)·{
    →  ListNode* p,·**next; →  //·메모리를·해제할·노드를·가리키는·포인터·p와·다음·노드를·가리키는·포인터·next

    →  p·=·head; →  →  →  //·p가·head부터·null까지·노드를·순회하며·메모리·해제
    →  while(p·!=·NULL)·{ →  //·p가·null·이·아닐·때까지·반복
    →  →  next·=·p->link; →  //·다음·노드·주소를·next에·저장
    →  →  free(p); →  //·p가·가리키는·노드의·메모리·해제
    →  →  p·=·next; →  →  //·삭제될·다음·노드·주소를·p의·값으로·변경
    →  →  }
}

```

13. 마지막으로 리스트의 모든 노드에 대한 메모리를 해제하는 clear 함수이다. ListNode 형 포인터 p 가 리스트 전체를 head 포인터가 가리키는 첫 노드부터 순회하며 각 노드의 메모리를 해제한다.

2.4 실행창

data.txt

data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) !

```
20172045 김지연 40 40 100
20170285 김준섭 60 20 85
20170488 전희원 80 40 20
```

결과 1

Microsoft Visual Studio 디버그 콘솔

```
<<성적 관리 프로그램>>
>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

2
=====
학 번 | 이 름 | 수학 | 영어 | 국어
=====
20172045 | 김지연 | 40 | 40 | 100
20170285 | 김준섭 | 60 | 20 | 85
20170488 | 전희원 | 80 | 40 | 20
=====

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

3
추가할 학생 이름 : 문준영
학번 : 20170544
수학 점수 : 50
영어 점수 : 30
국어 점수 : 70

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

2
=====
학 번 | 이 름 | 수학 | 영어 | 국어
=====
20172045 | 김지연 | 40 | 40 | 100
20170285 | 김준섭 | 60 | 20 | 85
20170488 | 전희원 | 80 | 40 | 20
20170544 | 문준영 | 50 | 30 | 70
=====
```

```
>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

1
검색할 학번 : 20170488
< 이름 : 전희원 >
< 수학 : 80점 >
< 영어 : 40점 >
< 국어 : 20점 >
< 총점 : 140점 >
< 평균 : 46점 >

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

0

C:\Users\이예빈\Desktop\CSE 2021-1\
료되었습니다(코드: -1073741819개).
이 창을 닫으려면 아무 키나 누르세요
```

결과 2

```

C:\ Microsoft Visual Studio 디버거 콘솔
<<성적 관리 프로그램>>
>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

3
추가할 학생 이름 : 이예빈
학번 : 20204059
수학 점수 : 100
영어 점수 : 98
국어 점수 : 93

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

2
=====
| 학 번 | 이 름 | 수학 | 영어 | 국어 |
=====
| 20172045 | 김지연 | 40 | 40 | 100 |
| 20170285 | 김주정 | 60 | 20 | 85 |
| 20170488 | 전희원 | 80 | 40 | 20 |
| 20204059 | 이예빈 | 100 | 98 | 93 |
=====

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

1
검색할 학번 : 20204058
학번 20204058 검색 실패

>> -0- 종료
>> -1- 검색
>> -2- 목록
>> -3- 추가

0

C:\Users\이예빈\Desktop\CSE_2021-1\과목들\자

```

2.5 느낀점

연결 리스트에 새로운 노드를 삽입하는 위치가 마지막이라는 점에서 어떻게 구현을 해야 할지 고민을 한 과제였다. 처음에는 원형 단순 연결 리스트로 구현을 하여 리스트의 마지막에 삽입을 하도록 구현을 하는 것을 고려해 보았다. 그러나 마지막 노드의 링크가 null이 아니므로 리스트 전체의 순회가 조금은 복잡하다는 점에서, 그리고 굳이 원형 리스트로 구현하지 않아도 된다는 점에서 원형 리스트로 자료를 구현하는 것은 보류하였다.

다음으로 고려해본 방법은 마지막 노드의 주소를 삽입 함수의 매개변수로 전달하는 방법이었다. 이 방법을 이용해 구현하는 과정에 있어서 헷갈렸던 점은 리스트를 가리키는 head 포인터와 포인터의 주소를 담은 이중 포인터를 사용하는 부분이었다. 보통 head 포인터의 값이 변경되는 경우 삽입 함수에서 변경된 head 포인터를 반환하는 방법으로 구현을 하였다. 그러나 마지막 노드를 가리키는 포인터 값도 새로운 노드가 삽입될 때마다 변경이 되기 때문에 이 포인터 값에 대한 고려 또한 하지 않을 수가 없었다. 결국 이중 포인터를 사용하여, 삽입 함수를 호출하는 곳에서는 head 포인터와 마지막 노드를 가리키는 포인터의 주소를 넘기고, 삽입 함수 내에서는 매개변수로 이중 포인터를 받는 방법을 선택하게 되었다.

3. 느낀점

두 실습 과제의 큰 차이점은 연결 리스트에서의 새로운 노드 삽입 위치였다. 1 번과 같은 경우는 버블 정렬을 구현할 때 많은 고민이 있었고, 2 번 과제의 경우 리스트의 끝에 노드를 삽입하기 위해 이중 포인터를 전달하는 부분에서 고민을 하였다.

이번 실습 과제들 모두 크게 어려움은 없었지만, 연결 리스트의 사용과 각 노드에 대한 접근, 그리고 각 노드들의 링크를 포인터와 이중 포인터를 사용하여 서로 연결을 시키는 부분에 대해 더 깊게 공부하고 고민함으로써 익숙해질 수 있도록 도와준 문제들이었다.

이제 2 주차 실습, 이론 과제를 끝으로 구조체와 포인터, 기본 연결 리스트에 대한 복습은 끝나겠지만 앞으로 배우고 구현하게 될 트리와 다양한 자료 구조들에 계속 사용하게 될 자료들이므로 충분히 더 연습하고 익숙해져 앞으로의 과제들을 푸는데 있어 막힘이 없도록 해야 되겠다.