

자료구조2 레포트



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

목 차

1. 파일 입출력, 동적 할당과 단순 연결리스트, 학번/이름/총점에 대한 정렬을 이용한
학생 성적 프로그램

1.1 문제 분석

1.2 소스 코드

1.3 소스 코드 분석

1.4 실행창

1.5 느낀점

파일 입출력, 동적 할당과 단순 연결리스트, 학번/이름/총점에 대한 정렬을 이용한 학생 정보 관리 프로그램

HW 1

- data.txt 파일에 학번, 이름, 총점이 저장 되어 있다. 이 정보를 동적 할당을 이용하여 단순 연결 리스트로 입력 받아 저장하고, 이 연결리스트를 이용하여 학번, 이름, 총점 순으로 정렬이 되도록 프로그램을 작성하여 제출 하시오. (9월 16일:목)
- 1. 학번으로 정렬: 선배 먼저
- 2. 이름으로 정렬: ㄱ 먼저
- 3. 총점으로 정렬: 높은 점수 먼저

1. 문제 분석

이 문제는 학번, 이름, 총점의 학생 데이터가 저장되어 있는 파일로부터 각 학생의 정보를 읽어 동적 할당한 학생 노드에 저장하고, 이를 단순 연결리스트에 삽입하고 학번, 이름, 총점에 따라 정렬을 하여 출력하는 프로그램이다. 이 프로그램을 작성하는데 있어 중심으로 고려해보아야 할 것들은 다음과 같다. (메모리 해제, 리스트 출력과 같이 이전 과제들에서 충분히 여러 번 반복하여 설명한 부분은 제외하였다.)

1. 구조체, 연결 리스트

- 학번, 이름(포인터), 총점을 멤버로 포함하는 학생 구조체
- 학생 구조체 변수와 자체 참조 구조체 포인터를 멤버로 하는 리스트의 노드

2. 파일 입출력, 그리고 동적 할당과 리스트에 노드 삽입

- 파일로부터 학생의 데이터를 입력 받을 때에는, 정수형인 학번과 총점은 임시 학생 구조체의 멤버 변수에 저장하면 되지만, 이름 문자열과 같은 경우는 포인터가 멤버로 선언되어 있으므로, 임시 문자열로 데이터를 받고 포인터에 문자열의 길이만큼 동적 할당하여 문자열을 해당 구조체의 이름 포인터가 가리키는 메모리에 복사하여야 한다.
- 이러한 과정을 위해서는, 파일에서 데이터를 읽는 반복문 내에서 문자열에 대한 메모리 할당, 그리고 새로운 학생 노드에 대한 메모리를 할당하여 생성된 노드를 리스트에 삽입하는 함수에 전달하여야 한다.

- 리스트에 노드를 삽입할 때에는, 리스트의 처음에 삽입하는 가장 구현이 쉬운 형태로 함수를 구현한다.

3. 버블 정렬

- 학번은 오름차순으로, 이름은 문자열의 사전식 오름차순으로, 총합은 내림차순으로 정렬을 구현한다. 데이터 n 개에 대하여 총 $(n-1)$ 번 노드를 순회하며 노드의 구조체 데이터 값을 교환하여 정렬을 완성하도록 한다. 한 번 순회를 돌 때마다, 가장 큰 또는 가장 작은 데이터 값이 가장 마지막의 노드의 데이터 필드에 저장된다. 아래 소스 코드 분석에 자세히 설명해 놓았지만, 각 순회가 끝날 때마다 다음 순회에서는 마지막 노드에 대해선 값을 비교하지 않아도 되므로, 이를 포인터를 이용하여 구현하였다.
- 문자열을 비교하는 경우에는 strcmp 함수를 사용하도록 한다. strcmp 의 반환값(음수, 0, 양수)을 고려하여야 한다.

2. 소스 코드

```
1  /*
2  → 작성자: .이예빈 (20204059)
3  → 작성일: .2021.09.15
4  → 프로그램명: .파일.입출력과.동적.할당, .연결.리스트를.이용하여.학생.정보를.저장하고,
5  → → → 리스트의.학생.데이터를.학번, .이름, .총점에.따라.정렬하는.프로그램
6  */
7
8  #include<stdio.h>
9  #include<stdlib.h>
10 #include<string.h>
11
12 // .학생.구조체
13 typedef struct student {
14 → // .학번, .이름, .총점
15 → int id;
16 → char* name; // .이름에.대한.문자열.동적.할당.받을.포인터
17 → int total;
18 } Student;
19
20 // .연결.리스트.노드.구조체
21 typedef struct ListNode {
22 → Student data;
23 → struct ListNode* link;
24 } ListNode;
25
26 ListNode* insert_first(ListNode* head, ListNode* new_node);
27 ListNode* sortBy_id(ListNode* head); → // .선배.먼저. (오름차순)
28 ListNode* sortBy_name(ListNode* head); → // .ㄱ.먼저. (오름차순)
29 ListNode* sortBy_total(ListNode* head); → // .높은.점수.먼저. (내림차순)
30 void print_list(ListNode* head);
31 void clear(ListNode* head);
32
33 int main() {
34 → FILE* fp = NULL;
35 → Student tmp; → → → // .파일로부터.읽은.학생.정보를.저장할.학생.구조체
36 → ListNode* head = NULL; → // .학생.정보를.담을.연결.리스트의.head.포인터
37 → char name[20]; → → → // .파일에서.읽는.학생의.이름.데이터를.저장할.문자열
38
39 → // .파일.열기.. (실패.코드.포함)
40 → fp = fopen("data.txt", "rt");
41 → if (fp == NULL) {
42 → → fprintf(stderr, ".파일.열기.실패");
43 → → exit(1);
44 → }
45
46 → // .학생.데이터.파일에서.읽으며.동적.할당.후.리스트에.삽입
47 → while (!feof(fp)) {
48 → → fscanf(fp, "%d%s%d", &tmp.id, name, &tmp.total);
49
50 → → ListNode* new_node = (ListNode*) malloc(sizeof(ListNode)); → // .새로운.노드.동적.할당
51 → → new_node->data = tmp; → // .새로운.노드의.데이터.필드에.파일에.읽은.tmp.값.저장(id, total)
52 → → new_node->data.name = (char*) malloc(sizeof(name)); → // .데이터.필드의.name.포인터에.메모리.동적.할당
53 → → if (new_node == NULL || new_node->data.name == NULL) { → // .노드, .또는.name.멤버의.동적.할당.오류시
54 → → → printf("동적.할당.오류.\n");
55 → → → exit(1);
56 → → }
57 → → strcpy(new_node->data.name, name); → // .새로운.노드.구조체의.name.메모리.영역에.name.문자열을.복사
58
59 → → head = insert_first(head, new_node); → // .리스트에.첫.노드로.삽입
60 → }
61 → printf("===== <.student.list> =====\n\n"); → → // .리스트.출력
62 → print_list(head);
63 }
```

```

64 → printf("\n===== <.sort.By.id> =====\n\n"); → // 학번·오름차순·정렬·후·출력
65 → head = sortBy_id(head);
66 → print_list(head);
67
68 → printf("\n===== <.sort.By.name> =====\n\n"); → // 이름·오름차순(사전식)·정렬·후·출력
69 → head = sortBy_name(head);
70 → print_list(head);
71
72 → printf("\n===== <.sort.By.total> =====\n\n"); → // 총점·내림차순·정렬·후·출력
73 → head = sortBy_total(head);
74 → print_list(head);
75
76 → clear(head); → // 메모리·해제
77 → fclose(fp); → // 파일·닫기
78 }
79
80 ListNode* insert_first(ListNode* head, ListNode* new_node) {
81 → new_node->link = head;
82 → head = new_node;
83 → return head;
84 }
85
86 ListNode* sortBy_id(ListNode* head) {
87 → ListNode* p = head, *q = NULL;
88 → Student tmp;
89 → int sorted;
90
91 → if (head == NULL) return; → // 빈·리스트의·경우
92 → do {
93 → → p = head; → → → → → → → // head·포인터에·대하여·정렬·반복
94 → → sorted = 0; → → → → → → → // 정렬·여부·확인·(초깃값=0)
95 → → while (p->link != q) { → → → → → // p의·다음·노드가·q일·때까지·반복하여·현재·노드와·다음·노드의·데이터·크기·비교
96 → → → // 다음·노드·데이터의·id값이·현재·노드의·것보다·큰·경우·두·구조체의·데이터를·교환(오름차순)
97 → → → if (p->data.id > p->link->data.id) {
98 → → → → tmp = p->data;
99 → → → → p->data = p->link->data;
100 → → → → p->link->data = tmp;
101 → → → → sorted = 1; → → → → // 한·번이라도·정렬이·된·경우,·값을·1로·변경
102 → → → → }
103 → → → p = p->link; → → → → // 다음·노드로·이동
104 → → → }
105 → → q = p; → → → → → → → // 순회를·마무리한·마무리·p·포인터·값을·q에·대입
106 → } while (sorted); → → → → // sorted가·1일·때까지·반복
107
108 → return head;
109 }
110
111 ListNode* sortBy_name(ListNode* head) {
112 → ListNode* p = head, *q = NULL;
113 → Student tmp;
114 → int sorted;
115
116 → if (head == NULL) return; → // 빈·리스트의·경우
117 → do {
118 → → p = head;
119 → → sorted = 0;
120 → → while (p->link != q) {
121 → → → // strcmp·함수를·이용해·다음·노드의·문자열의·순서가·앞에·온다면·두·구조체의·데이터를·교환(사전식·오름차순)
122 → → → if (strcmp(p->data.name, p->link->data.name) > 0) {
123 → → → → tmp = p->data;

```

```

124     →     →     →     → p->data.=p->link->data;
125     →     →     →     → p->link->data.=tmp;
126     →     →     →     → sorted.=1;
127     →     →     →     → }
128     →     →     →     → p.=p->link;
129     →     →     →     → }
130     →     →     →     → q.=p;
131     →     →     →     → }.while.(sorted);
132
133     →     return head;
134 }
135
136 ListNode*.sortBy_total(ListNode*.head)}.{
137     →     ListNode*.p.=head,.*.q.=NULL;
138     →     Student tmp;
139     →     int sorted;
140
141     →     if.(head==NULL).return;
142     →     do.{
143     →     →     p.=head;
144     →     →     sorted.=0;
145     →     →     while.(p->link.!=q)}.{
146     →     →     →     → // 다음 노드 데이터의 total 멤버 값이 현재 노드의 것보다 작은 경우 두 구조체의 데이터값을 변경 (내림차순)
147     →     →     →     → if.(p->data.total.<p->link->data.total)}.{
148     →     →     →     →     tmp.=p->data;
149     →     →     →     →     p->data.=p->link->data;
150     →     →     →     →     p->link->data.=tmp;
151     →     →     →     →     sorted.=1;
152     →     →     →     → }
153     →     →     →     → p.=p->link;
154     →     →     →     → }
155     →     →     →     → }.while.(sorted);
156     →     return head;
157 }
158
159 void print_list(ListNode*.head)}.{
160     →     ListNode*.p.=head;
161     →     while.(p.!=NULL)}.{
162     →     →     printf("%d.%s.%d\n", p->data.id, p->data.name, p->data.total);
163     →     →     p.=p->link;
164     →     →     }
165     →     }
166
167 void clear(ListNode*.head)}.{
168     →     ListNode*.p,.*.next;
169
170     →     p.=head;
171     →     while.(p.!=NULL)}.{
172     →     →     next.=p->link;
173     →     →     free(p);
174     →     →     p.=next;
175     →     →     }
176     →     }0

```

3. 소스 코드 분석

```
/*
→ 작성자: .이예빈 (20204059)
→ 작성일: .2021.09.15
→ 프로그래밍: .파일.입출력과.동적.할당, .연결.리스트를.이용하여.학생.정보를.저장하고,
→   →   →   리스트의.학생.데이터를.학번, .이름, .총점에.따라.정렬하는.프로그램
*/

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

//.학생.구조체
typedef struct student{
→   //.학번, .이름, .총점
→   int id;
→   char* name; //.이름에.대한.문자열.동적.할당.받을.포인터
→   int total;
}Student;

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

1. 주석에 작성자, 작성일, 프로그램명을 작성하고, 필요한 헤더 파일들을 추가한다. 문자열을 비교하는 strcpy 함수를 사용하는 string 헤더 파일까지 추가하였다.
2. 학번 id, 이름 문자열을 가리키는 문자 포인터 name, 총점 total 을 멤버로 하는 학생 구조체 Student, 그리고 연결 리스트의 노드 구조체를 선언한다.

```
ListNode* insert_first(ListNode* head, ListNode* new_node);
ListNode* sortBy_id(ListNode* head); →   //.선배.먼저.(오름차순)
ListNode* sortBy_name(ListNode* head); →   //.ㄱ.먼저.(오름차순)
ListNode* sortBy_total(ListNode* head); →   //.높은.점수.먼저.(내림차순)
void print_list(ListNode* head);
void clear(ListNode* head);
```

3. 함수 원형을 선언한다. 학생 리스트의 head 포인터와 새로운 학생 데이터를 담은 노드의 포인터를 매개 변수로 하여 리스트의 첫 노드에 삽입하는 insert_first, 학번을 오름차순으로 정렬하는 sortBy_id, 이름 문자열을 오름차순 사전식으로 정렬하는 sortBy_name, 총점을 내림차순으로 정렬하는 sortBy_total, 그리고 리스트의 전체 데이터를 출력하는 print_list 와 리스트의 모든 노드의 메모리를 해제하는 clear 함수이다.


```

int main() {
    FILE* fp = NULL;
    Student tmp; // 파일로부터 읽은 학생 정보를 저장할 학생 구조체
    ListNode* head = NULL; // 학생 정보를 담은 연결 리스트의 head 포인터
    char name[20]; // 파일에서 읽은 학생의 이름 데이터를 저장할 문자열

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

```

4. main 함수가 시작된다. 파일 포인터 fp, 파일로부터 읽은 학생 정보를 임시로 저장할 학생 구조체 tmp, 학생 리스트의 head 포인터, 그리고 파일로부터 읽을 때 학생의 이름을 임시로 저장할 문자배열 name 을 선언한다. 문자열의 길이는 넉넉히 20 으로 설정하였다.

5. 이후 "data.txt" 파일을 읽기 모드로 연다. 파일 열기를 실패한 경우의 코드도 포함하였다.

```

// 학생 데이터 파일에서 읽으며, 동적 할당 후 리스트에 삽입
while (!feof(fp)) {
    fscanf(fp, "%d%s%d", &tmp.id, name, &tmp.total);

    ListNode* new_node = (ListNode*) malloc(sizeof(ListNode)); // 새로운 노드 동적 할당
    new_node->data = tmp; // 새로운 노드의 데이터 필드에 파일에 읽은 tmp 값 저장 (id, total)
    new_node->data.name = (char*) malloc(sizeof(name)); // 데이터 필드의 name 포인터에 메모리 동적 할당
    if (new_node == NULL || new_node->data.name == NULL) { // 노드, 또는 name 멤버의 동적 할당 오류시
        printf("동적 할당 오류\n");
        exit(1);
    }

    strcpy(new_node->data.name, name); // 새로운 노드 구조체의 name 메모리 영역에 name 문자열을 복사

    head = insert_first(head, new_node); // 리스트에 첫 노드로 삽입
}

```

6. 파일로부터 학생의 데이터를 읽는다. 먼저 한 줄씩 읽을 때마다, 임시 구조체 tmp 에는 학번과 총점인 id 와 total 값을 저장하고, 문자열과 같은 경우 구조체에는 포인터형으로 name 이 선언된 상태이기 때문에 우선적으로 char 문자배열인 name 에 저장한다.

7. 새로운 학생 데이터를 저장할 노드 new_node 포인터에 메모리를 동적 할당하고, 이 노드의 데이터 필드에 tmp 구조체를 우선적으로 대입한다. 다음으로 데이터 필드의 name 문자형 포인터에 파일로부터 읽은 이름 name 의 길이만큼 메모리를 할당하고, 포인터에 대해 동적 할당 오류가 나지 않는 경우를 확인하도록 한다. 이후에 노드의 name 데이터 필드에 name 문자열을 복사하여 새로운 학생 노드를 완성한다.

8. 마지막으로, 생성된 노드를 head 포인터와 함께 insert_first 함수의 인수로 전달하여 리스트의 첫 노드로 삽입하도록 한다. head 포인터가 가리키는 값이 계속해서 변경되므로 해당 함수에서 head 포인터를 반환한 값으로 다시 head 를 변경하도록 한다.

```

→ printf("=====<.student.list.>=====\n\n"); → // .리스트 .출력
→ print_list(head);

→ printf("\n=====<.sort.By.id.>=====\n\n"); → // .학번 .오름차순 .정렬 .후 .출력
→ head.=.sortBy_id(head);
→ print_list(head);

→ printf("\n=====<.sort.By.name.>=====\n\n"); → // .이름 .오름차순 (사전식) .정렬 .후 .출력
→ head.=.sortBy_name(head);
→ print_list(head);

→ printf("\n=====<.sort.By.total.>=====\n\n"); → // .총점 .내림차순 .정렬 .후 .출력
→ head.=.sortBy_total(head);
→ print_list(head);

→ clear(head); → // .메모리 .해제
→ fclose(fp); → // .파일 .닫기
}

```

10. 리스트를 출력한다. 우선 정렬 전, 파일로부터 읽은 모든 학생 데이터를 저장한 리스트를 출력해보도록 한다. 이후 학번 오름차순, 이름 오름차순(사전식), 총점 내림차순 정렬을 하고 정렬되어 변경된 리스트를 각각 출력하여 확인하도록 한다. 각 함수의 구현은 아래 설명을 통해 확인하도록 하자.

11. 마지막에는 clear 함수를 호출하여 리스트의 모든 노드에 대한 메모리를 해제한다. 파일을 닫고 프로그램 실행을 종료하도록 한다.

이제 각 함수를 살펴보도록 하자.

```

ListNode*.insert_first(ListNode*.head, ListNode*.new_node){
→ new_node->link.=.head;
→ head.=.new_node;
→ return head;
}

```

12. main 함수에서 파일로부터 각 학생의 데이터를 읽어 동적 할당된 메모리에 저장한 새로운 학생 노드를 가리키는 포인터, 그리고 리스트의 head 포인터를 매개변수로 받아 리스트의 처음에 삽입하는 insert_first 함수이다. 새로운 노드의 링크 필드에 head 값을 대입하고, 다시 head 에는 새로운 노드의 주소를 대입하여 이전의 첫 노드와 링크를 연결하도록 한다. 마지막으로 변경된 head 를 다시 반환하게 된다.

다음으로는 학번, 이름, 총점에 따라 정렬을 구현한 함수들을 살펴보자. 모두 동일한 버블 정렬이 사용되었고, 구현에 있어서 조금씩 수정된 부분만 있기 때문에 긴 설명을 반복하지는 않겠다.

```

ListNode* sortBy_id(ListNode* head) {
    → ListNode* p = head, *q = NULL;
    → Student tmp;
    → int sorted;

    → 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
            → // 다음 노드 데이터의 id 값이 현재 노드의 것보다 큰 경우 두 구조체의 데이터를 교환 (오름차순)
            → if (p->data.id > p->link->data.id) {
                → 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 마무리 p 포인터 값을 q에 대입
    } while (sorted); // sorted가 1일 때까지 반복

    → return head;
}

```

13. 버블 정렬을 이용하여 학번을 오름차순으로 정렬한 sortBy_id 함수이다. ListNode의 포인터형으로 선언된 p가 리스트를 순회하기를 반복하며 버블 정렬이 구현된다. 포인터 p는 동일한 포인터형인 q의 이전 노드까지 리스트를 순회하는데, NULL 값으로 처음 초기화되어 있는 q까지 순회한다는 것은, 첫 순회 때에는 p가 리스트의 마지막인 NULL 이전의 노드까지 도달하여 순회를 하며 구조체 값을 교환한다는 의미이다.

do~while 반복문 내에서 0으로 초기화된 sorted의 값이 1로 변경되는 경우에 반복문을 계속 실행하도록 한다. 이때 sorted가 1로 변경된다는 것은 오름차순 정렬을 위해 두 노드의 데이터 값이 한 번이라도 변경이 되었다는 뜻이다. 두 값의 교환이 더 이상 일어나지 않는다는 것은 리스트의 전체 노드에 대한 정렬이 완성되었다는 뜻이므로 반복문은 실행을 종료하게 된다.

처음 순회할 때 p는 NULL 값을 가지는 q의 이전, 즉 리스트의 마지막 노드까지 순회하게 되는데, 안쪽 반복문이 종료되고 나서 q는 p의 값을 갖게 된다. 결국 q는 리스트의 끝을 의미하는 NULL 값부터 시작해 왼쪽 방향으로 차례대로 이동하며 각 노드를 가리키게 되는 것이다. 이를 통해 한 번의 순회 때마다 가장 큰 값을 가진 데이터가 마지막 노드가 된다는 점을 이용해 순회를 하게 될 때마다 반복의 횟수가 1씩 줄어든다.

```

ListNode* sortBy_name(ListNode* head) {
    ListNode* p = head, *q = NULL;
    Student tmp;
    int sorted;

    if (head == NULL) return; // 빈 리스트의 경우
    do {
        p = head;
        sorted = 0;
        while (p->link != q) {
            // strcmp 함수를 이용해 다음 노드의 문자열의 순서가 앞에 온다면 두 구조체의 데이터를 교환 (사전식 오름차순)
            if (strcmp(p->data.name, p->link->data.name) > 0) {
                tmp = p->data;
                p->data = p->link->data;
                p->link->data = tmp;
                sorted = 1;
            }
            p = p->link;
        }
        q = p;
    } while (sorted);

    return head;
}

```

14. 학생 노드의 name 데이터 필드 값을 비교하여 사전식 오름차순으로 정렬하는 sortBy_name 함수이다. 버블 정렬은 이전의 함수와 구현 방식이 동일하다. 다만 두 문자열을 비교할 때 strcmp 함수를 사용하여 현재 노드의 name 문자열이 다음 노드의 것보다 사전식 순서가 뒤에 있다면 두 구조체 데이터를 변경하도록 한 것이 차이점이다.

```

ListNode* sortBy_total(ListNode* head) {
    ListNode* p = head, *q = NULL;
    Student tmp;
    int sorted;

    if (head == NULL) return;
    do {
        p = head;
        sorted = 0;
        while (p->link != q) {
            // 다음 노드 데이터의 total 멤버 값이 현재 노드의 것보다 작은 경우 두 구조체의 데이터 값을 변경 (내림차순)
            if (p->data.total < p->link->data.total) {
                tmp = p->data;
                p->data = p->link->data;
                p->link->data = tmp;
                sorted = 1;
            }
            p = p->link;
        }
        q = p;
    } while (sorted);

    return head;
}

```

15. 학생 노드의 total 데이터 필드 값을 비교하여 내림차순으로 정렬하는 sortBy_total 함수이다. 내림차순이라는 점만 다르고, 버블 정렬의 구현은 이전의 것들과 동일하다.

```

void print_list(ListNode* head) {
    ListNode* p = head;
    while (p != NULL) {
        printf("%d %s %d\n", p->data.id, p->data.name, p->data.total);
        p = p->link;
    }
}

void clear(ListNode* head) {
    ListNode* p, *next;


    p = head;
    while (p != NULL) {
        next = p->link;
        free(p);
        p = next;
    }
}

```

16. 마지막으로 리스트 전체를 출력하는 `print_list`, 그리고 리스트의 모든 노드에 대한 메모리를 해제하는 `clear` 함수이다. 각각 포인터가 리스트의 헤드포인터가 가리키는 첫 노드부터 순회하며 데이터를 출력하고, 또는 해당 노드의 메모리를 해제한다.

4. 실행창

data.txt

 data.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V)


20201111 이예빈 92

20190000 홍길동 91

20170000 김유신 82

20171234 유관순 87

결과창

 Microsoft Visual Studio 디버그 콘솔

===== < student list > =====

20171234 유관순 87

20170000 김유신 82

20190000 홍길동 91

20201111 이예빈 92

===== < sort By id > =====

20170000 김유신 82

20171234 유관순 87

20190000 홍길동 91

20201111 이예빈 92

===== < sort By name > =====

20170000 김유신 82

20171234 유관순 87

20201111 이예빈 92

20190000 홍길동 91

===== < sort By total > =====

20201111 이예빈 92

20190000 홍길동 91

20171234 유관순 87

20170000 김유신 82

C:\Users\이예빈\Desktop\CSE 2021-2\자료구조2

.

5. 느낀점

이전의 실습 과제를 하면서 이미 버블 정렬을 구현하는 부분에서 고민을 많이 했기 때문에, 이번 이론 과제를 하면서 큰 어려움은 없었다. 다만 이전 과제에서는 노드의 데이터 필드 부분을 함수의 매개변수로 전달하여 해당 함수에서 동적 할당을 하여 리스트에 삽입하였다. 반면, 이번 과제에서는 구조체의 이름 포인터형에 문자열에 대한 동적 할당이 필요했기 때문에, main 함수 내에서 노드에 대한 데이터 할당, 그리고 문자열에 대한 데이터 할당을 하고, 만들어진 완성된 노드를 함수로 전달하여 해당 함수 내에서 리스트에 노드를 삽입하는 방식으로 구현을 하였다.

이전에는 char 문자열에 대해서 직접 배열을 선언하여 구조체의 멤버로 사용했다면, 앞으로는 문자열에 대한 메모리 또한 동적으로 할당을 하게 된다. 동적 할당을 하고 문자열을 복사하는 과정이 추가되기 때문에 이전보다 복잡해지긴 했지만, 나중에 길이의 차이가 서로 다른 문자열들을 저장해야 하는 경우에는 메모리 공간을 효율적으로 사용할 수 있게 된다는 점에서는 장점이 되지 않을까 싶다.

이제는 메모리를 동적 할당하고 포인터에 메모리의 주소를 대입하는 과정이 이전보다 훨씬 익숙해졌다. 이러한 익숙함을 바탕으로, 앞으로 배우게 될 새로운 자료들에 대해서도 리스트와 포인터를 잘 다루어야 되겠다.