Data Structure1 Report

순천향대학교

과목명 | 자료구조1

담당교수 | 홍 민 교수님

학과 | 컴퓨터소프트웨어공학과

학년 | 2학년

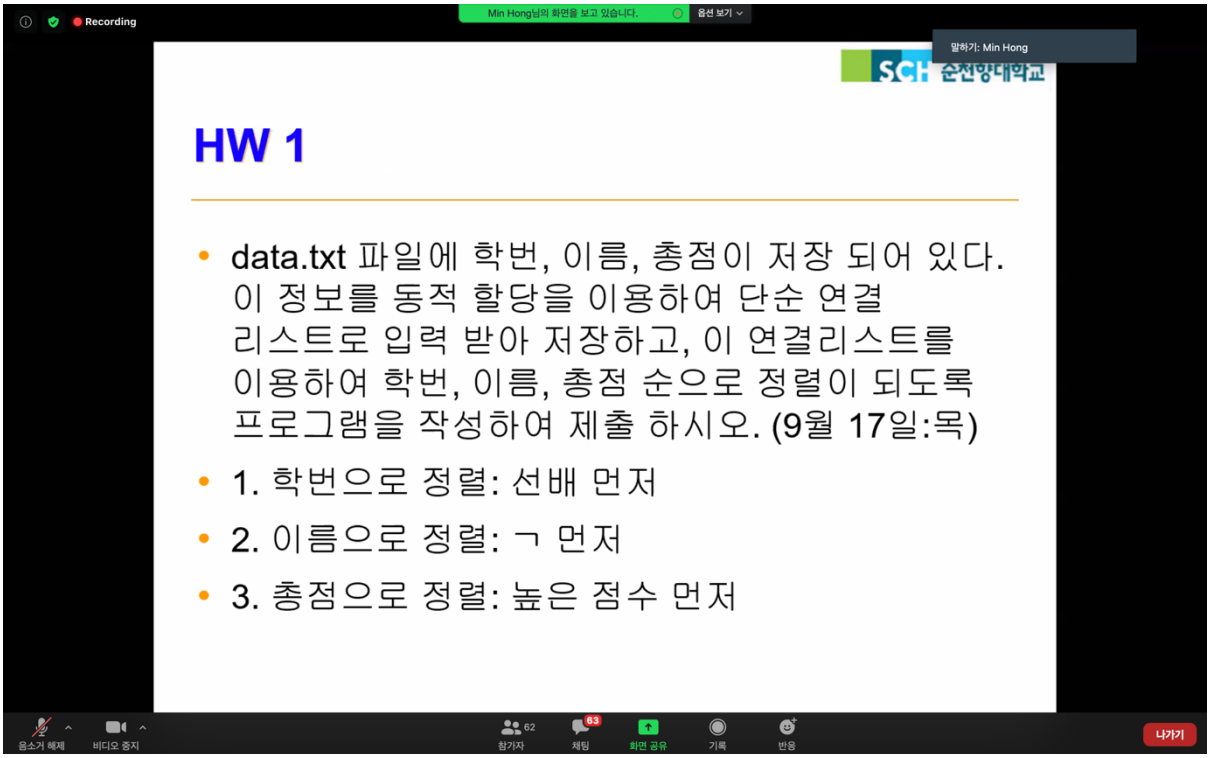
학번 | 20194059

이름 | 김태완

제출일 | 2020.09.17

목차

1. 연결리스트를 이용해 이름, 학번, 총점 순으로 정렬하는 프로그램



1.1 문제 분석

이번 과제는 data.txt 에 들어있는 학생의 학번, 이름, 총점에 대한 정보를 파일 입출력을 통해 불러온 후 단순 연결리스트에 이 데이터들을 저장해 버블 정렬을 통해 학번, 이름, 총점 순서대로 정렬하는 문제이다. 이미 저번 학기 자료구조 1 수업을 통해 data.txt 에서 데이터를 불러오는 법, 연결리스트에 데이터를 삽입 하는 것은 많이 해 보았기 때문에 저번 학기에 공부를 열심히 했으면 여기까지는 크게 어려운 것은 아닌 것 같다. 하지만 이것들을 학번, 이름, 총점에 대해 각각 버블 정렬 알고리즘을 적용하여 정렬 하는 방법은 아직 배우지 않아서 이 부분만 잘 해결하면 어쩌면 쉽게 풀 수 있는 문제 일 것 같다. 하지만 정렬을 하는 알고리즘에 대한 것은 구글이나 깃허브 등 인터넷에 약간만 찾아보면 정말 많은 코드들이 있기 때문에 이것들을 참고 해서 나만의 코드를 작성 하면 될 것 같다.

1.2 소스코드

```c
//자료구조2_HW1
//작성자 : 20194059 김태완
//작성일 : 2020.09.11
//프로그램명 : 연결리스트를 이용해 이름, 학번, 총점 순으로 정렬하는 프로그램

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

typedef struct student
// 학생의 정보를 담고있는 구조체
{
    int num; // 학번
    char name[30]; // 이름
    float grade; // 총점
}STUDENT;

typedef struct node
// 단일 링크드리스트 노드
{
    STUDENT student; // 데이터필드
    struct node *link; // 링크필드
}NODE;

//함수 원형 정의
void error(char* string);
NODE* insert_data(NODE* head, STUDENT value);
void print_type(char *string);
void print_list(NODE *head);
void free_memory(NODE* head);
NODE* sort_by_number(NODE* head);
NODE* sort_by_name(NODE* head);
NODE* sort_by_grade(NODE *head);

int main( )
{
    FILE *fp; // 파일 포인터 생성
    STUDENT tmp; // 파일에서 데이터를 받아오기 위해 임시로 사용하는 변수
    NODE* head = NULL; // 연결리스트의 맨 처음 노드를 가리키는 변수


    fp = fopen("data.txt","r"); // 파일 오픈

    if(fp == NULL) //파일 오픈 오류시 에러 출력 후 코드 종료
    {
        error("file open error");
    }

    while(!feof(fp))
    {
        fscanf(fp,"%d %s %f",&tmp.num,tmp.name,&tmp.grade);
        // 파일에서 학번, 이름, 총점 순으로 데이터를 읽어온다
        head = insert_data(head, tmp);
        // 연결리스트에 파일로부터 불러온 데이터를 삽입한다
    }
    rewind(fp); // 파일 포인터의 위치를 맨 앞으로 돌린다
    fclose(fp); // 파일 닫기

    // 정렬 전 데이터 출력
    print_type("   정렬 전    ");
    print_list(head); // 노드를 옮겨가며 데이터 출력
    printf("₩n");

    print_type("학번 기준 정렬");
    head = sort_by_number(head); // 학번을 기준으로 데이터 정렬
    print_list(head); // 노드를 옮겨가며 데이터 출력
    printf("₩n");

```

```
    print_type("이름 기준 정렬");
    head = sort_by_name(head); // 이름을 기준으로 데이터 정렬
    print_list(head); // 노드를 옮겨가며 데이터 출력
    printf("₩n");

    print_type("총점 기준 정렬");
    head = sort_by_grade(head); // 총점을 기준으로 데이터 정렬
    print_list(head); // 노드를 옮겨가며 데이터 출력

    free_memory(head); // 메모리 해제
    return 0; //프로그램 종료
}

void error(char* string)
// 에러를 출력하고 프로그램을 종료하는 함수
{
    printf("%s₩n",string); // 매개변수로 받아온 문자열 (오류문장) 출력
    exit(1);
}

NODE* insert_data(NODE* head, STUDENT value)
// 연결리스트에 데이터를 삽입하는 함수
{
    NODE *p = (NODE *)malloc(sizeof(NODE)); // 메모리 동적 할당
    if(p==NULL) // 동적 할당 오류 처리
    {
        error("memory allocate arror");
    }
    p->student = value; // 데이터필드에 데이터 대입
    p->link = head; // 원래 헤드의 주소를 p의 링크 필드에 대입
    // 만약 헤드가 비어있어도 메인 함수에서 헤드를 NULL로 초기화 해 주어 맨 마지막 p->link에
    // NULL 값이 들어갈 채로 새로운 데이터가 채워지기 때문에 따로 헤드가 비어있을때의 처리를 안 해주어도 된다
    head = p; // 헤드를 p로 변경
    return head; // 헤드포인터의 주소 반환
}

void print_type(char *string)
// 리스트 출력 전 어떤 방식으로 정렬했는지 알려주기 위한 출력 함수
{
    printf("┌──────────────────────────────┐₩n");
    printf("|       %8s      |₩n",string); // 매개변수로 받아온 문자열 출력
    printf("└──────────────────────────────┘₩n");
}

void print_list(NODE *head)
// 리스트를 출력하는 함수
{
    NODE *p = NULL;
    printf("┌──────────┬──────────┬──────┐₩n");
    printf("|    학번   |    이름  | 총점  |₩n");
    printf("├──────────┼──────────┼──────┤₩n");

    for(p = head;p!=NULL;p=p->link) // 노드가 끝날 때 까지 반복
    {
        printf("| %8d | %4s | %.2f |₩n",p->student.num,p->student.name,p->student.grade);
        // 학번 이름 총점 순으로 출력
    }
    printf("└──────────┴──────────┴──────┘₩n");
}

void free_memory(NODE* head)
// 연결리스트에 사용된 메모리를 해제하는 함수
{
    NODE *p = NULL;
    NODE *tmp = NULL;
    for(p = head;p!=NULL;p=p->link) // 노드가 끝날 때 까지 반복
    {
        tmp = p; // tmp에 p의 주소를 대입 해 주며
        free(tmp); // tmp에 담긴 주소의 메모리를 해제
    }
}
```

```
142 void swap(NODE* n1, NODE* n2)
143 // 두 노드의 데이터를 서로 바꿔주는 함수
144 {
145     STUDENT tmp;
146     // 두 데이터의 위치를 바꿔준다
147     tmp = n1->student;
148     n1->student = n2->student;
149     n2->student=tmp;
150 }
151
152 NODE* sort_by_number(NODE* head)
153 // 학번을 기준으로 정렬하는 함수
154 {
155     NODE* p = NULL;
156     NODE* k = NULL;
157
158     // 버블정렬 알고리즘 (학번은 오름차순으로 정렬 해야함)
159     for(p=head;p->link!=NULL;p=p->link)
160     {
161         for(k=head;k->link!=NULL;k=k->link)
162         {
163             if(k->student.num > k->link->student.num) // 앞에있는 학번이 더 클 경우
164             {
165                 swap(k,k->link); // 두 데이터의 위치 변경
166             }
167         }
168     }
169     return head; // 헤드 포인터 반환
170 }
171
172 NODE* sort_by_name(NODE* head)
173 // 이름을 기준으로 정렬하는 함수
174 {
175     NODE* p = NULL;
176     NODE* k = NULL;
177
178     // 버블정렬 알고리즘 (이름은 사전순 오름차순으로 정렬)
179     for(p=head;p->link!=NULL;p=p->link)
180     {
181         for(k=head;k->link!=NULL;k=k->link)
182         {
183             if(strcmp(k->student.name,k->link->student.name)>0)
184             // 앞에 있는 데이터가 사전순으로 더 클 경우
185             {
186                 swap(k,k->link); // 두 데이터의 위치 변성
187             }
188         }
189     }
190     return head; // 헤드 포인터 반환
191 }
192
193 NODE* sort_by_grade(NODE *head)
194 // 점수를 기준으로 정렬하는 함수
195 {
196     NODE* p = NULL;
197     NODE* k = NULL;
198
199     // 버블 정렬 알고리즘 (총점은 내림차순 정렬)
200     for(p=head;p->link!=NULL;p=p->link)
201     {
202         for(k=head;k->link!=NULL;k=k->link)
203         {
204             if(k->link->student.grade > k->student.grade) // 뒤에 있는 데이터가 더 클 경우
205             {
206                 swap(k,k->link); // 두 데이터의 위치 변경
207             }
208         }
209     }
210     return head; // 헤드 포인터 반환
211 }
212
```

1.3 소스코드 분석

```
1  //자료구조2_HW1
2  //작성자 : 20194059 김태완
3  //작성일 : 2020.09.11
4  //프로그램명 : 연결리스트를 이용해 이름, 학번, 총점 순으로 정렬하는 프로그램
5
6  #define _CRT_SECURE_NO_WARNINGS
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 typedef struct student
12 // 학생의 정보를 담고있는 구조체
13 {
14     int num; // 학번
15     char name[30]; // 이름
16     float grade; // 총점
17 }STUDENT;
18
19 typedef struct node
20 // 단일 링크드리스트 노드
21 {
22     STUDENT student; // 데이터필드
23     struct node *link; // 링크필드
24 }NODE;
25
```

1. 프로그램의 정보를 담고 있는 주석을 작성한다.
2. 프로그램 작성에 필요한 헤더 파일을 포함한다.
3. 학생의 정보를 담고 있는 student 구조체를 정의한다.
4. 단일 연결리스트의 노드를 정의한다.

```
26 //함수 원형 정의
27 void error(char* string);
28 NODE* insert_data(NODE* head, STUDENT value);
29 void print_type(char *string);
30 void print_list(NODE *head);
31 void free_memory(NODE* head);
32 NODE* sort_by_number(NODE* head);
33 NODE* sort_by_name(NODE* head);
34 NODE* sort_by_grade(NODE *head);
35
```

5. 밑에서 정의 해 둔 함수들의 원형을 정의한다.

```
83 void error(char* string)
84 // 에러를 출력하고 프로그램을 종료하는 함수
85 {
86     printf("%s₩n",string); // 매개변수로 받아온 문자열 (오류문장) 출력
87     exit(1);
88 }
89
```

6. 매개변수로 받아온 문자열 (오류 문장) 을 출력하고 프로그램을 종료시키는 error 함수를 정의한다.

```c
NODE* insert_data(NODE* head, STUDENT value)
// 연결리스트에 데이터를 삽입하는 함수
{
    NODE *p = (NODE *)malloc(sizeof(NODE)); // 메모리 동적 할당
    if(p==NULL) // 동적 할당 오류 처리
    {
        error("memory allocate arror");
    }
    p->student = value; // 데이터필드에 데이터 대입
    p->link = head; // 원래 헤드의 주소를 p의 링크 필드에 대입
    // 만약 헤드가 비어있어도 메인 함수에서 헤드를 NULL로 초기화 해 주어 맨 마지막 p->link에
    // NULL 값이 들어갈 채로 새로운 데이터가 채워지기 때문에 따로 헤드가 비어있을때의 처리를 안 해주어도 된다
    head = p; // 헤드를 p로 변경
    return head; // 헤드포인터의 주소 반환
}
```

7. 연결리스트에 새로운 데이터를 삽입하는 함수를 정의한다.

이때 새로운 노드를 새로운 헤드 포인터로 설정하고 이 전의 헤드 포인터를 새로 만들어진 노드의 링크 필드에 저장 해 주는 방식을 사용 하였다.

이 경우 메인함수에서 head 포인터 생성시 NULL 로 초기화 하였고 먼저 들어온 데이터가 가장 마지막 데이터가 되는 스택과 비슷한 구조로 되어 있기 때문에 가장 처음에 생성된 p->link 에는 NULL 값이 들어간 채로 새로운 노드가 이 노드의 앞으로 차례차례 채워지게 된다 따라서 헤드 포인터가 비어있을 경우에 대한 처리를 따로 해 주지 않아도 된다.

```c
void print_type(char *string)
// 리스트 출력 전 어떤 방식으로 정렬했는지 알려주기 위한 출력 함수
{
    printf("┌──────────────────────────────┐₩n");
    printf("|       %8s      |₩n",string); // 매개변수로 받아온 문자열 출력
    printf("└──────────────────────────────┘₩n");
}
```

8. 메인함수로부터 문자열을 받아와 정렬 전 / 이름 기준 정렬 / 학번 기준 정렬 등 어떤 방식으로 출력되는지를 알려주는 함수를 정의 해준다.

```c
void print_list(NODE *head)
// 리스트를 출력하는 함수
{
    NODE *p = NULL;
    printf("┌────────────┬──────────┬────────┐₩n");
    printf("|    학번    |    이름  | 총점  |₩n");
    printf("├────────────┼──────────┼────────┤₩n");

    for(p = head;p!=NULL;p=p->link) // 노드가 끝날 때 까지 반복
    {
        printf("| %8d | %4s | %.2f |₩n",p->student.num,p->student.name,p->student.grade);
        // 학번 이름 총점 순으로 출력
    }
    printf("└────────────┴──────────┴────────┘₩n");
}
```

9. 출력 형식과 함께 연결 리스트에 저장된 데이터를 학번, 이름, 총점 순으로 출력 해 주는 함수를 정의한다.

```
130 void free_memory(NODE* head)
131 // 연결리스트에 사용된 메모리를 해제하는 함수
132 {
133     NODE *p = NULL;
134     NODE *tmp = NULL;
135     for(p = head;p!=NULL;p=p->link) // 노드가 끝날 때 까지 반복
136     {
137         tmp = p; // tmp에 p의 주소를 대입 해 주며
138         free(tmp); // tmp에 담긴 주소의 메모리를 해제
139     }
140 }
141
```

10. 연결리스트를 위해 할당된 메모리를 노드 하나하나 돌아가며 메모리를 반납 해주는 함수를 정의한다.

```
142 void swap(NODE* n1, NODE* n2)
143 // 두 노드의 데이터를 서로 바꿔주는 함수
144 {
145     STUDENT tmp;
146     // 두 데이터의 위치를 바꿔준다
147     tmp = n1->student;
148     n1->student = n2->student;
149     n2->student=tmp;
150 }
151
```

11. 버블 정렬 알고리즘을 사용 하기 위해 서로 위치를 바꿀 두 노드의 주소를 매개변수로 받아와 두 노드의 데이터를 서로 바꿔주는 함수 이다.

```
152 NODE* sort_by_number(NODE* head)
153 // 학번을 기준으로 정렬하는 함수
154 {
155     NODE* p = NULL;
156     NODE* k = NULL;
157
158     // 버블정렬 알고리즘 (학번은 오름차순으로 정렬 해야함)
159     for(p=head;p->link!=NULL;p=p->link)
160     {
161         for(k=head;k->link!=NULL;k=k->link)
162         {
163             if(k->student.num > k->link->student.num) // 앞에있는 학번이 더 클 경우
164             {
165                 swap(k,k->link); // 두 데이터의 위치 변경
166             }
167         }
168     }
169     return head; // 헤드 포인터 반환
170 }
171
```

12. 학번을 기준으로 리스트를 정렬해주는 함수이다.

정렬은 보두 버블 정렬 방식을 사용 하였고 학번의 경우 선배가 먼저 출력 되어야 하므로 숫자가 낮은 학번부터 높은 학번까지 이어지는 오름차순으로 정렬 하였다.

```
NODE* sort_by_name(NODE* head)
// 이름을 기준으로 정렬하는 함수
{
    NODE* p = NULL;
    NODE* k = NULL;

    // 버블정렬 알고리즘 (이름은 사전순 오름차순으로 정렬)
    for(p=head;p->link!=NULL;p=p->link)
    {
        for(k=head;k->link!=NULL;k=k->link)
        {
            if(strcmp(k->student.name,k->link->student.name)>0)
            // 앞에 있는 데이터가 사전순으로 더 클 경우
            {
                swap(k,k->link); // 두 데이터의 위치 변경
            }
        }
    }
    return head; // 헤드 포인터 반환
}

```

13. 이름을 기준으로 리스트를 정렬 해주는 함수이다.

이름의 경우 ㄱ,ㄴ,ㄷ 즉 사전순으로 정렬 되어야 하므로 학번과 strcmp() 함수를 사용하여 두 문자열을 비교 한 후 오름차순으로 정렬 하였다.

```
NODE* sort_by_grade(NODE *head)
// 점수를 기준으로 정렬하는 함수
{
    NODE* p = NULL;
    NODE* k = NULL;

    // 버블 정렬 알고리즘 (총점은 내림차순 정렬)
    for(p=head;p->link!=NULL;p=p->link)
    {
        for(k=head;k->link!=NULL;k=k->link)
        {
            if(k->link->student.grade > k->student.grade) // 뒤에 있는 데이터가 더 클 경우
            {
                swap(k,k->link); // 두 데이터의 위치 변경
            }
        }
    }
    return head; // 헤드 포인터 반환
}

```

14. 총점을 기준으로 정렬 해주는 함수로 총점의 경우 높은 점수부터 낮은 점수 순으로 출력 되어야 하기 때문에 내림차순으로 정렬 하였다.

```
36 int main()
37 {
38     FILE *fp; // 파일 포인터 생성
39     STUDENT tmp; // 파일에서 데이터를 받아오기 위해 임시로 사용하는 변수
40     NODE* head = NULL; // 연결리스트의 맨 처음 노드를 가리키는 변수
41
42
43     fp = fopen("data.txt","r"); // 파일 오픈
44
45     if(fp == NULL) //파일 오픈 오류시 에러 출력 후 코드 종료
46     {
47         error("file open error");
48     }
49
```

15. 파일 포인터 변수 fp, 파일에서 받아온 데이터를 임시 저장해줄 tmp, 연결리스트의 헤드 부분의 주소를 담고 있는 변수 head 를 선언한다.

16. 파일을 읽기 전용 모드로 연다.

17. 파일을 열 수 없을 시 "file open error" 이라는 오류를 출력 하고 프로그램을 종료시킨다.

```
50     while(!feof(fp))
51     {
52         fscanf(fp,"%d %s %f",&tmp.num,tmp.name,&tmp.grade);
53         // 파일에서 학번, 이름, 총점 순으로 데이터를 읽어온다
54         head = insert_data(head, tmp);
55         // 연결리스트에 파일로부터 불러온 데이터를 삽입한다
56     }
57     rewind(fp); // 파일 포인터의 위치를 맨 앞으로 돌린다
58     fclose(fp); // 파일 닫기
59
```

18. 파일의 끝까지 반복시키며 data.txt 에 저장된 학생의 학번, 이름, 총점의 정보를 읽어와 연결리스트에 읽어온 정보를 삽입한다.

19. 파일 포인터의 위치를 맨 앞으로 돌린 후 파일을 닫는다.

```
60     // 정렬 전 데이터 출력
61     print_type("   정렬 전    ");
62     print_list(head); // 노드를 옮겨가며 데이터 출력
63     printf("₩n");
64
```

20. 정렬 전의 리스트를 출력한다.

```
65      print_type("학번 기준 정렬");
66      head = sort_by_number(head); // 학번을 기준으로 데이터 정렬
67      print_list(head); // 노드를 옮겨가며 데이터 출력
68      printf("₩n");
69
```

21. 학번을 기준으로 리스트를 정렬하고 정렬 결과를 출력한다.

```
70      print_type("이름 기준 정렬");
71      head = sort_by_name(head); // 이름을 기준으로 데이터 정렬
72      print_list(head); // 노드를 옮겨가며 데이터 출력
73      printf("₩n");
74
```

22. 이름을 기준으로 리스트를 정렬하고 정렬 결과를 출력한다.

```
75      print_type("총점 기준 정렬");
76      head = sort_by_grade(head); // 총점을 기준으로 데이터 정렬
77      print_list(head); // 노드를 옮겨가며 데이터 출력
78
79      free_memory(head); // 메모리 해제
80      return 0; //프로그램 종료
81  }
82
```

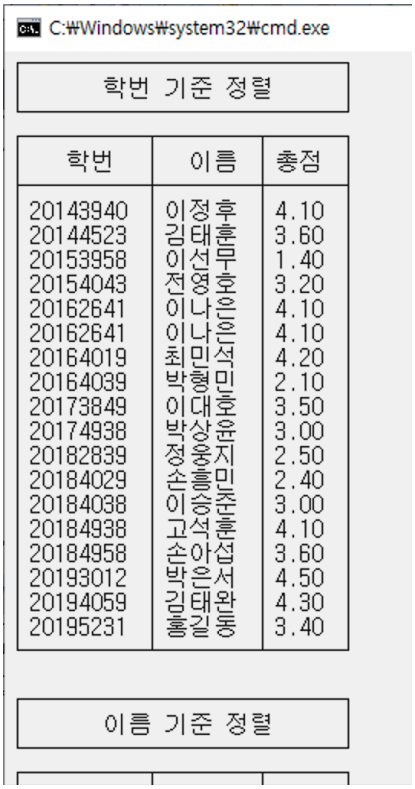
23. 총점을 기준으로 리스트를 정렬하고 그 결과를 출력한다.

24. 연결리스트에 할당된 메모리를 반납한다.

25. 프로그램을 종료한다.

1.4 실행창

1. data.txt 정렬 전 출력



2. data.txt 학번 기준 정렬 후 출력

3. data.txt 이름 기준 정렬 후 출력

4. data.txt 총점 기준 정렬 후 출력

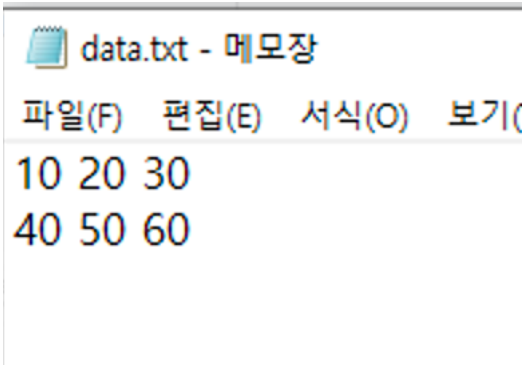
1.5 느낀점

앞서 이 문제를 분석 할 때에도 이야기 했지만, 이 문제에서 가장 중요 한 점은 파일을 읽어 오는 것, 연결리스트를 이용 하는 것 보다 버블 정렬을 연결리스트에 적용 하고, 그것을 학번, 이름, 총점 즉 정수형, 문자열, 실수형 총 세가지의 자료형으로 정렬 할 수 있는 것이 가장 중요 하다고 생각 했고 거기서 가장 많은 시간을 쓴 것 같다. 버블 정렬을 해야 한다고 생각 하고 가장 처음 든 생각은 swap()함수를 구현 할 때 노드의 위치를 아예 바꿔버리는 방법과, 안에 들어있는 데이터 필드의 값들만 바꾸는 방법 이 두가지를 생각했다. 먼저 시도한 방법은 노드 자체의 위치를 바꾸는 것 이였다.

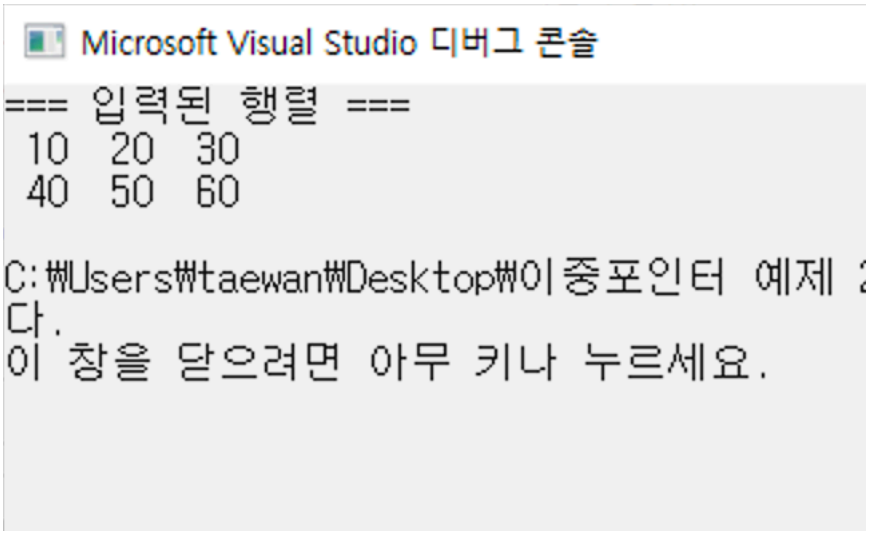
```
80  NODE* sort_by_number(NODE* head)
81  // 학번을 기준으로 정렬하는 함수
82  {
83      NODE* p = NULL;
84      NODE* k = NULL;
85      NODE* tmp = NULL;
86
87      for(p=head;p!=NULL;p=p->link)
88      {
89          for(k=p;k!=NULL;k=k->link)
90          {
91              if(k->link->student.num > k->student.num)
92              {
93                  if(k==p)
94                  {
95                      k->link = k->link->link;
96                      k->link->link = k;
97                  }
98                  else
99                  {
100                     //k->link = tmp->link;
101                     k->link = k->link->link;
102                     k->link->link = k;
103                     //k->link = k;
104                     tmp->link = k->link;
105                 }
106             }
107             tmp = k;
108         }
109         tmp = p;
110     }
111     return head;
112 }
113
```

위의 코드는 내 깃허브의 커밋 기록중 노드의 위치를 바꾸는 것을 시도 해 보았을 때의 코드이다 노드 자체의 자리를 바꾸려면 이 전 노드의 주소를 알고 있어야 한다는 것을 알았고, 단순 연결리스트로는 이전 노드의 주소를 바로 알 수 없기 때문에 tmp 라는 새로운 변수를 사용해보기도 하였지만 어떻게 코드를 작성 해 보아도 계속 오류가 나 결국 이 방법은 포기하고 데이터만 교환하는 형식으로 방법을 바꾸게 되었다. 구현은 훨씬 쉽게 하였지만 뭔가 2 프로 만족스럽지 못했다.
따라서 다음 번에 다시 기회가 된다면 단순 연결 리스트가 아닌 이중 연결 리스트를 이용 하여 노드 자체의 위치를 바꾸는 방식으로 버블 정렬 알고리즘을 다시 작성 해보고 싶다.

2. 이중 포인터 예제2 오류 찾기



오류가 있는 부분은 소스코드 상에 있는게 아니라 data.txt 파일 안에 있었다.
기존 교수님께서 올려주신 data.txt 파일에는 커서가 40 50 60 이 적힌 줄의
바로 아래 줄을 가리키고 있었다. 하지만 이렇게 되면 반복문에 !feof(fp)
라는 조건을 주었기 때문에 60 까지 읽어도 파일의 끝 임을 감지하지 못해
한번 더 반복이 되지만 이때는 아무런 데이터가 저장 되어 있지 않기 때문에
쓰레기 값이 들어가게 되어 수업 때와 같은 오류가 발생 했던 것이다.
따라서 커서의 위치를 60 의 바로 오른쪽으로 옮기고 저장을 한 후 실행을
다시 하게 되면



위와 같이 정상적인 출력을 얻을 수 있게 된다.

3. 느낀점

2 학기가 시작되고 처음 받는 자료구조 과제였다. 방학 동안 웹과 관련된 공부를 하느라 c 언어는 거의 쳐다 보지도 않고 파이썬 언어를 주로 하다 보니 c 언어 코드를 작성 하는 도중 if 옆에 괄호를 치지 않는 실수, 조건, 반복문을 쓸때 중괄호를 쓰는게 아닌 :을 붙이는 실수, 세미콜론을 붙이지 않는 실수 등을 많이 반복하며 사람은 정말 적응의 동물이고 난 방학 동안 씨언어는 거의 하지 않았구나 라는 후회를 하기도 했다. 또한 교수님께서 첫 주와 두번째 주 까지 포인터, 구조체, 동적 할당 등 c 언어 복습을 해 주시지 않았더라면 문법 마저도 다 까먹을 뻔 했다. 1 학기때 자료구조 1 성적을 생각보다 너무 잘 받아서 이정도는 안해도 되겠지 하고 방심 했던게 정말 컸던 것 같다. 이번 기회에 c 언어 문법에 대해 다시 익혔고 앞으로 내가 어떤 다른 언어를 공부 하던 내가 원래 할 수 있는 언어를 이용해 자주 코드를 작성 해 보며 그 언어의 문법을 까먹지 않게 하는 것도 굉장히 중요 하다는 것을 느꼈다. 다시 새로운 학기가 시작 되었고 자료 구조 수업도 다시 시작 되었으니 1 학기때의 성적보다 더 좋은 자료구조 성적을 받을 수 있도록 더욱 노력 해야겠다.