Algorithm Report

-학생 정보 정렬 프로그램_HW1-

제 출 일	2024.09.26	전 공	컴퓨터소프트웨어공학과
과 목	알고리즘	학 번	20233523
담당교수	홍 민	이 름	최윤호

목 차

1. 문제 제시

- data.txt 파일에 학번, 이름, 총점이 저장 되어 있다. 이 정보를 동적 할당을 이용하여 교재에서 제공된 단순 연결 리스트 함수를 이용하여 데이터를 입력 받아 저장하고, 이 연결리스트를 이용하여 학번, 이름, 총점 순으로 버블정렬로 정렬이 되도록 프로그램을 작성하여 제출 하시오. (9월 26일: 목)
- 학생이름은 최소 메모리를 사용하여 저장
- 학번으로 정렬: 선배 먼저
- 이름으로 정렬: ㄱ 먼저

총점으로 정렬: 높은 점수 먼저

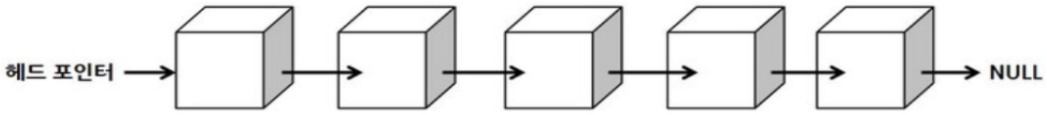
2. 문제 분석

2.1 버블 정렬

먼저 버블 정렬에 대해 알아야 한다. 버블정렬이란 정렬 알고리즘의 일종으로 인접한두 개의 레코드를 서로 비교하고 정렬되지 않은 경우 정렬을 시킨다 이러한 방식으로 데이터 개수 n만큼 반복하면 오름차순 정렬이라 가정할 때 가장 큰 값이 가장 마지막에 위치하게 되며 정렬이 끝날 때까지 반복하면 정렬이 끝나게 된다. 이러한 방식이 거품이 떠오르는 것 같아 하여 부쳐진 이름이다.

2.2 연결리스트를 활용한 버블정렬

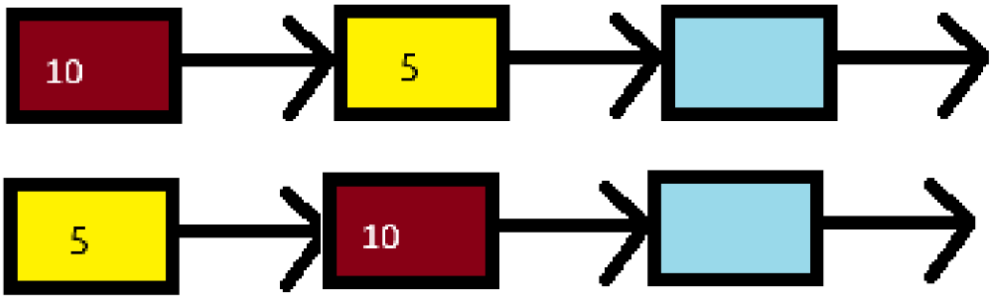
연결리스트의 구조를 간략히 설명하자면 아래의 사진과 같다.



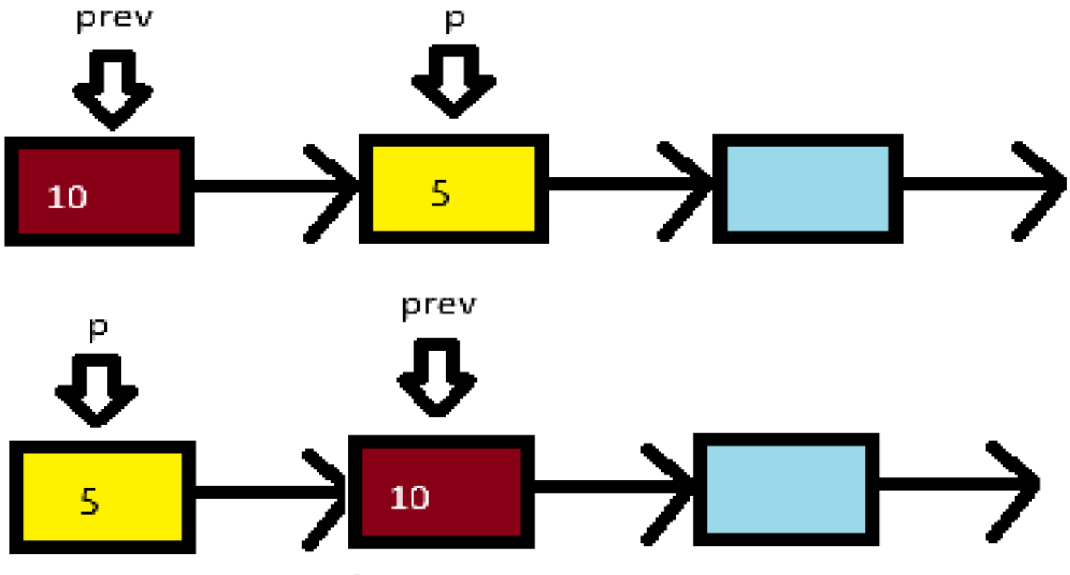
노드들이 서로 연결되어 있는 방식이고 노드는 데이터와 다음 노드를 가르키는 포인터로 구성되어있다.그럼 연결리스트를 이용한 버블정렬의 과정을 보자, 오름차순 정렬이라고 가정한다.

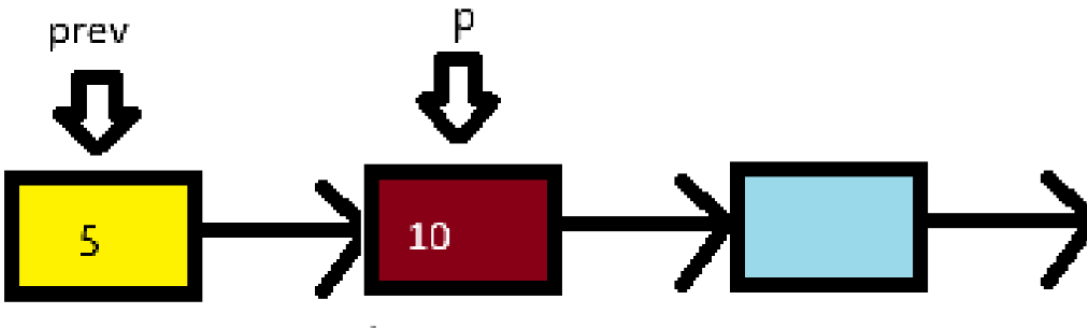
2.2.1 첫번째 노드를 비교하는 경우

첫 번째 노드의 값을 비교하는 경우를 보자.



이와 같이 노드 속 데이터만 정렬되는 것이 아닌 노드의 위치 자체가 변경되어야 한다. 그러기 위해서는 노드를 가리키는 포인터들을 설정해야 하는데 비교를 하고자 하는 노드를 가리키는 p, p의 직전 노드를 가리키는 prev 노드가 필요하다. 하지만 첫 번째 노드를 비교하기 위해서는 prev노드를 생성할 수 없으므로 첫 번째 노드를 비교하는 경우에는 prev를 p라고 생각하고 연산하여야 한다. 그 후 prev와 p를 서로 뒤바꿔 두 번째 비교부터는 정상적으로 사용되게 해야 한다.아래와 같다.



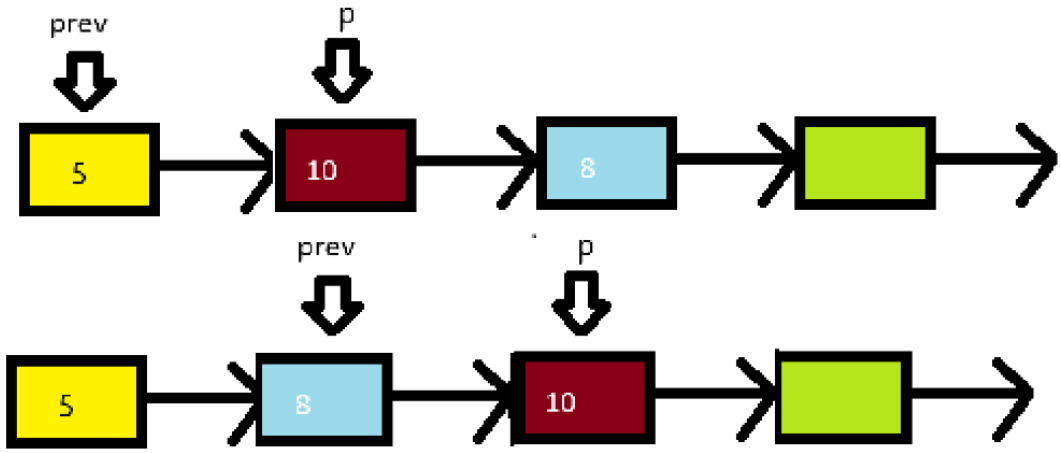


이 방식을 위해서는 아래의 연산이 필요하다.

```
if (prev>p)
{
      prev->link = p->link;
      p->link = prev;
      prev = p;
      p = p->link;
}
```

2.2.2 첫 번째 노드가 아닌경우

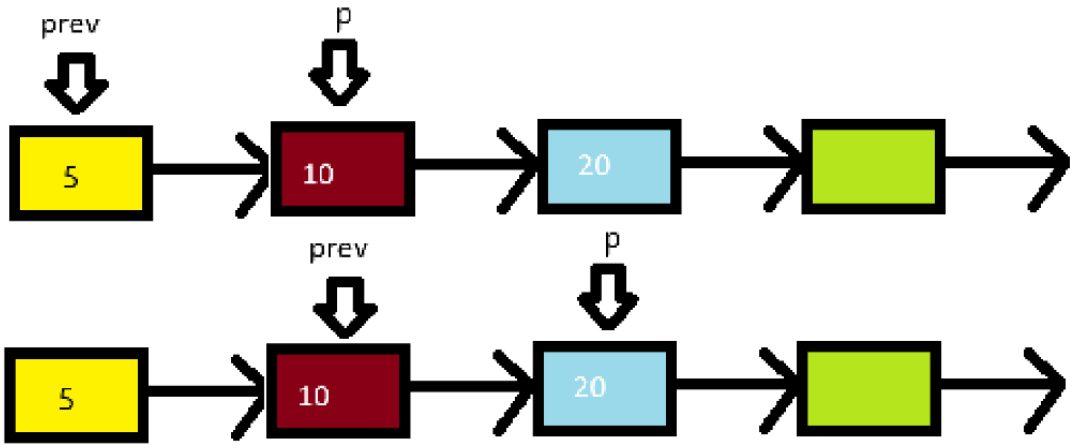
첫 번째 노드가 아닌 경우는 prev와 p를 의도한 바대로 사용하면 된다. p의 다음 노드의 데이터보다 p의 데이터가 더 크다면 서로의 위치를 뒤바꾼다. 하지만 다음 노드의 데이터가 더 크다면 p가 다른 노드를 가리키게 만든다. 다음은 p의 다음 노드보다 p의 값이 더 큰 경우의 과정을 타나 낸다.



이 방식을 위해서는 아래의 연산이 필요하다.

```
if (p>p.link)
{
      prev->link = p->link;
      p->link = p->link->link;
      prev->link->link = p;
      prev = prev->link;
}
```

그럼 이제 p의 다음 노드의 값이 더 큰 경우를 보자




```
else
{
            p = p->link;
            prev = prev->link;
}
```

이러한 방식으로 p가 리스트 끝(n-1번 째)에 온다면 1차 탐색이 종료되고, 다시 처음으로 돌아가 다시 반복해 이미 정렬된 노드를 제외한 n-2 번째꺼지 반복 ...n-3 번째까지 반복하면 최종적인 오름차순 정렬이 완료된다. 이러한 방식은 최악의 경우 시간복잡도가 O(n^2)이다.

3. 전체 코드

뒷장에 첨부.

```c
/*
작성일 : 2024_09_25
작성자 : 최윤호
프로그램 명 : 학생정보 정렬 프로그램
*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//학생정보를 담을 데이터 구조체
typedef struct element {
    int stu_num;
    int score;
    char* name;
}element;

//노드 구조체
typedef struct ListNode {   // 노드 타입
    element data;
    struct ListNode* link;
} ListNode;


ListNode* insert_first(ListNode* head,element data)
{
    ListNode* p = (ListNode*)malloc(sizeof(ListNode));
    p->data = data;
    p->link = head;
    head = p;
    return head;
}

//연결리스트의 노드 개수를 반환하는 함수
int Count_Node(ListNode* head)
{
    ListNode* temp = head;
    int count = 1;
    for (int i = 0; temp->link != NULL; i++)
    {
        temp = temp->link;
        count++;
    }

    return count;
}

void print_list(ListNode* head)
{
    printf("%s₩t%12s%24s₩n₩n", "학번","이름","성적");
    for (ListNode* p = head; p != NULL; p = p->link)
        printf("%d₩t%-20s%7d₩n", p->data.stu_num, p->data.name, p->data.score);
}

//학번 순 버블정렬 함수
ListNode* Sort_Stunum(ListNode* head,int Count)
```

```
57  {
58      ListNode* p;//비교할 데이터를 가르키는 포인터
59      ListNode* prev;//비교할 데이터의 전 인덱스를 가르키는 포인터
60      ListNode* fir = head;//노드 순서에 맞게 head포인터를 새로고침하기 위한 포인터
61
62      //내림차순으로 정렬 (~후배(학번이 낮음))
63      for (int i = 0; i < Count - 1 ; i++)//데이터 n-1번반복
64      {
65          p = fir->link;//비교할 데이터를 맨 처음 데이터의 다음 데이터로 지정
66          prev = fir;//비교할 데이터의 전 인덱스 데이터를 지정
67          for (int j = 0; j < Count - i-1; j++)//n-i-1번 반복
68          {
69
70              //첫번째 노드인경우
71              if (j == 0)
72              {
73                  //첫번째 노드의 경우 실질적인 첫 노드는 prev의 학번이므로 prev에 저장➥
                       학번값을 비교
74                  if ((prev)->data.stu_num > (p)->data.stu_num)//prev의 학번이 실질적인➥
                       2번째 인덱스 p의 학번보다 작으면
75                  {
76                      prev->link = p->link; //3번째 노드를 첫번째 노드인 prev의 링크에 ➥
                         연결
77                      p->link = prev;//2번째 노드인 p를 첫번째 노드로 만들기 위에 1번째➥
                         노드인 prev앞으로 오도록 연결
78                      prev = p;//prev에 1번째 노드로 바뀐 p를 대입
79                      p = p->link;//2번째 노드를 가르키게 하기위해 1번째 노드로 바뀐 p ➥
                         의 링크로 이동
80                      fir = prev;//변경 후 1번째 노드인 prev를 fir에 저장;
81                  }
82              }
83              else
84              {
85                  if (p->data.stu_num > p->link->data.stu_num)
86                  {
87                      prev->link = p->link;//p노드의 전 노드를 p노드의 다음노드 바로 전➥
                         에 오도록 연결
88                      p->link = p->link->link;//p노드가 p->link 노드에 위치해야 하기때 ➥
                         문에 p노드를 p->link->link 바로 전에 오도록 연결
89                      prev->link->link = p;//원래 p의 다음노드였던 노드를 p의 바로 전에➥
                         연결
90                      prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞➥
                         으로 이동
91                  }
92                  else
93                  {
94                      p = p->link;//p다음 노드가 p보다 큰 경우이기 때문에 비교할 데이터➥
                         를 p->link로 대입
95                      prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞➥
                         으로 이동
96                  }
97              }
98          }
99      }
100
101     return fir;//최종적인 head노드를 반환
```

```
102 }
103
104
105 //총점순 버블정렬 함수
106 ListNode* Sort_Score(ListNode* head, int Count)
107 {
108     ListNode* p;//비교할 데이터를 가르키는 포인터
109     ListNode* prev;//비교할 데이터의 전 인덱스를 가르키는 포인터
110     ListNode* fir = head;//노드 순서에 맞게 head포인터를 새로고침하기 위한 포인터
111
112     //내림차순으로 정렬 (~낮은 성적)
113     for (int i = 0; i < Count - 1; i++)//데이터 n-1번반복
114     {
115         p = fir->link;//비교할 데이터를 맨 처음 데이터의 다음 데이터로 지정
116         prev = fir;//비교할 데이터의 전 인덱스 데이터를 지정
117
118         for (int j = 0; j < Count - i-1; j++)//n-i-1번 반복
119         {
120
121             //첫번째 노드인경우
122             if (j == 0)
123             {
124                 //첫번째 노드의 경우 실질적인 첫 노드는 prev에 저장된 성적이므로 prev
                      에 저장된 성적값을 비교
125                 if (prev->data.score < p->data.score)
126                 {
127                     prev->link = p->link; //3번째 노드를 첫번째 노드인 prev의 링크에
                          연결
128                     p->link = prev;//2번째 노드인 p를 첫번째 노드로 만들기 위에 1번째
                          노드인 prev앞으로 오도록 연결
129                     prev = p;//prev에 1번째 노드로 바뀐 p를 대입
130                     p = p->link;//2번째 노드를 가르키게 하기위해 1번째 노드로 바뀐 p
                          의 링크로 이동
131                     fir = prev;//변경 후 1번째 노드인 prev를 fir에 저장;
132                 }
133             }
134             else
135             {
136                 if ( p->data.score < p->link->data.score)//비교할 데이터를 담고있는
                      노드 p의 성적이 다음 노드의 학번보다 작은 경우
137                 {
138
139                     prev->link = p->link;//p노드의 전 노드를 p노드의 다음노드 바로 전
                          에 오도록 연결
140                     p->link = p->link->link;//p노드가 p->link 노드에 위치해야 하기때
                          문에 p노드를 p->link->link 바로 전에 오도록 연결
141                     prev->link->link = p;//원래 p의 다음노드였던 노드를 p의 바로 전에
                          연결
142                     prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞
                          으로 이동
143                 }
144                 else
145                 {
146                     p = p->link;//p다음 노드의 학번이 p의 성적 보다 작은 경우이기 때
                          문에 비교할 데이터를 p->link로 대입
147                     prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞
```

```
                    으로 이동
148                 }
149             }
150         }
151     }
152
153     return fir;//최종적인 head노드를 반환
154 }
155
156 //이름 순 버블정렬 함수
157 ListNode* Sort_Name(ListNode* head, int Count)
158 {
159     ListNode* p;//비교할 데이터를 가르키는 포인터
160     ListNode* prev;//비교할 데이터의 전 인덱스를 가르키는 포인터
161     ListNode* fir = head;//노드 순서에 맞게 head포인터를 새로고침하기 위한 포인터
162
163     //오름차순 (~ㅎ) 으로 정렬
164     for (int i = 0; i < Count - 1; i++)//데이터 n-1번반복
165     {
166         p = fir->link;//비교할 데이터를 맨 처음 데이터의 다음 데이터로 지정
167         prev = fir;//비교할 데이터의 전 인덱스 데이터를 지정
168         for (int j = 0; j < Count - i-1; j++)//n-i-1번 반복
169         {
170
171             //첫번째 노드인경우
172             if (j == 0)
173             {
174                 //첫번째 노드의 경우 실질적인 첫 노드는 prev변수이므로 prev에 저장된 ➥
                    값을 비교
175                 if (strcmp((prev)->data.name , (p)->data.name)>0)//strcmp함수를 이용 ➥
                    해 prev의 값이 실질적인 2번째 인덱스 p보다 크면
176                 {
177                     prev->link = p->link; //3번째 노드를 첫번째 노드인 prev의 링크에 ➥
                        연결
178                     p->link = prev;//2번째 노드인 p를 첫번째 노드로 만들기 위에 1번째➥
                        노드인 prev앞으로 오도록 연결
179                     prev = p;//prev에 1번째 노드로 바뀐 p를 대입
180                     p = p->link;//2번째 노드를 가르키게 하기위해 1번째 노드로 바뀐 p ➥
                        의 링크로 이동
181                     fir = prev;//변경 후 1번째 노드인 prev를 fir에 저장;
182                 }
183             }
184             else
185             {
186                 if ((strcmp(p->data.name , p->link->data.name) >0))//비교할 데이터를 ➥
                    담고있는 노드 p가 다음 노드보다 큰 경우
187                 {
188                     prev->link = p->link;//p노드의 전 노드를 p노드의 다음노드 바로 전➥
                        에 오도록 연결
189                     p->link = p->link->link;//p노드가 p->link 노드에 위치해야 하기때 ➥
                        문에 p노드를 p->link->link 바로 전에 오도록 연결
190                     prev->link->link = p;//원래 p의 다음노드였던 노드를 p의 바로 전에➥
                        연결
191                     prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞➥
                        으로 이동
192                 }
```

```
193                 else//그렇지 않은경우
194                 {
195                     p = p->link;//p다음 노드가 p보다 큰 경우이기 때문에 비교할 데이터를 p->link로 대입
196                     prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞으로 이동
197                 }
198             }
199         }
200     }
201
202     return fir;//최종적인 head노드를 반환
203 }
204
205 //동적할당 해제 함수
206 void node_free(ListNode* head)
207 {
208     while (head!=NULL)
209     {
210         ListNode* remove=head;
211         head = head->link;
212         free(remove->data.name);
213         free(remove);
214     }
215 }
216
217
218 // 테스트 프로그램
219 int main(void)
220 {
221     ListNode* head = NULL;
222
223     FILE* fp;
224     fp=fopen("data.txt", "r");
225     while (!feof(fp))
226     {
227         element data;
228         char name[100];
229         fscanf(fp,"%d %s %d", &data.stu_num, name, &data.score);
230         data.name = (char*)malloc(strlen(name) + 1);
231         strcpy(data.name, name);
232         //연결리스트에 추가
233         head=insert_first(head,data);
234     }
235
236     //node개수 탐색
237     int Count = Count_Node(head);
238     //학번순 정렬 후 출력
239     head = Sort_Stunum(head, Count);
240     printf("============================================================₩n₩t₩t학번순 정렬 (오름차순)₩n============================================================₩n₩n");
241     print_list(head);
242
243     //성적순 정렬 후 출력
244     head = Sort_Score(head, Count);
```

```
245     printf("₩n=============================================================₩n₩t₩t성적순
           정렬 (내림차순)₩n=============================================================₩n
          ₩n");
246     print_list(head);
247
248     //이름순 정렬 후 출력
249     head = Sort_Name(head, Count);
250     printf("₩n=============================================================₩n₩t₩t이름순
           정렬 (ㄱ ~ ㅎ)₩n=============================================================₩n
          ₩n");
251     print_list(head);
252
253     //동적할당 메모리 해제
254     node_free(head);
255
256     return 0;
257 }
```

4. 코드 분석

```
11 //학생정보를 담을 데이터 구조체
12 typedef struct element {
13     int stu_num;
14     int score;
15     char* name;
16 }element;
17
18 //노드 구조체
19 typedef struct ListNode {   // 노드 타입
20     element data;
21     struct ListNode* link;
22 } ListNode;
```

학생 정도를 저장하는 데이터 구조체를 element로 선언 후 연결리스트를 위한 노드 구조체를 생성해 준다.

```
25 ListNode* insert_first(ListNode* head,element data)
26 {
27     ListNode* p = (ListNode*)malloc(sizeof(ListNode));
28     p->data = data;
29     p->link = head;
30     head = p;
31     return head;
32 }
```

노드를 추가하는 함수로 첫 번째 노드의 바로 직전에 추가하는 insert_first함수이다. 헤드 포인터가 갱신된다.

```
34 //연결리스트의 노드 개수를 반환하는 함수
35 int Count_Node(ListNode* head)
36 {
37     ListNode* temp = head;
38     int count = 1;
39     for (int i = 0; temp->link != NULL; i++)
40     {
41         temp = temp->link;
42         count++;
43     }
44
45     return count;
```

연결 리스트의 노드 개수를 리턴하는 함수로 버블 정렬에서 반복 횟수를 제어하기 위해 필요하다. head 포인터가 변하지 않도록 temp라는 노드 포인터를 생성해 temp를 이용해 카운트한다.

버블 정렬를 진행하는 함수를 소개하겠다. 학번 순은 오름차순으로 정렬해야 선배부터 출력되며 성적은 내림차순으로 정렬해야 고득점자부터 출력된다. 두 함수에서의 차이점은 값을 비교하는 변수명과 부등호의 방향만 다르다. 또한 이름 정렬의 경우 아스키코드를 이용해 대소를 비교하는 "strcmp()"함수를 사용해 정렬을 시킨다. 점수 정렬은 ("data.stu_num"->"data.score"),('>' -> '<') 로 변경하면 점수를 정렬하는 함수이며 이름 정렬의 경우 if 문의 조건식을 첫 번째 노드를 정렬하는 조건식의 경우 "strcmp((prev)->data.name , (p)->data.name)>0" 로 그 이외 노드의 정렬은 "(strcmp(p->data.name , p->link->data.name) >0)" 로 조건식을 수정시키면 구할 수 있다. 변수와 조건식 변경 외 로직은 동일하기 때문에 코드 분석에서는 학번 순 정렬 함수를 예로 들어 분석하겠다.

```
55  //학번 순 버블정렬 함수
56  ListNode* Sort_Stunum(ListNode* head,int Count)
```

함수를 선언하고 매개변수 헤드 포인터와 노드의 개수를 전달받는다.

```
58      ListNode* p;//비교할 데이터를 가르키는 포인터
59      ListNode* prev;//비교할 데이터의 전 인덱스를 가르키는 포인터
60      ListNode* fir = head;//노드 순서에 맞게 head포인터를 새로고침하기 위한 포인터
```

비교를 할 대상이 되는 노드를 가리키는 포인터 p, p의 직전 노드를 가리키는 prev 포인터, head를 갱신시키기위해 1번째 노드를 저장시킬 fir 포인터를 선언한다.

```
63      for (int i = 0; i < Count - 1 ; i++)//데이터 n-1번반복
```

정렬을 시키기 위한 비교식의 반복이 한번 끝나면 하나의 노드가 정렬된 이기 때문에 모든 노드를 정렬시키기 위해 n-1 번 반복한다.

```
65          p = fir->link;//비교할 데이터를 맨 처음 데이터의 다음 데이터로 지정
66          prev = fir;//비교할 데이터의 전 인덱스 데이터를 지정
```

n-1 차 비교 전 i가 증가할 때마다 비교 대상인 2번째 노드를 p에, 그 직전 노드를 prev에 저장시킨다.

```
67          for (int j = 0; j < Count - i-1; j++)//n-i-1번 반복
```

하나에 원소를 정렬시키기 위해 n-1 번 반복하나, 정렬 후 정렬된 원소를 비교하지 않기 위해 n-1-i번으로 제어해 i가 증가할 때마다 끝에서부터 비교하는 노드를 줄인다.

```
70              //첫번째 노드인경우
71              if (j == 0)
72              {
73                  //첫번째 노드의 경우 실질적인 첫 노드는 prev의 학번이므로 prev에 저장➥
                       학번값을 비교
74                  if ((prev)->data.stu_num > (p)->data.stu_num)//prev의 학번이 실질적인➥
                       2번째 인덱스 p의 학번보다 작으면
75                  {
76                      prev->link = p->link; //3번째 노드를 첫번째 노드인 prev의 링크에 ➥
                         연결
77                      p->link = prev;//2번째 노드인 p를 첫번째 노드로 만들기 위에 1번째➥
                           노드인 prev앞으로 오도록 연결
78                      prev = p;//prev에 1번째 노드로 바뀐 p를 대입
79                      p = p->link;//2번째 노드를 가르키게 하기위해 1번째 노드로 바뀐 p ➥
                         의 링크로 이동
80                      fir = prev;//변경 후 1번째 노드인 prev를 fir에 저장;
81                  }
82              }
```

첫 번째 노드인 경우 사실상 prev 포인터의 노드를 비교해야 하기 때문에 p포인터와 비교한다. 만약 값이 크다면 첫 노드인 prev의 다음 노드를 3번째 노드인 p의 링크로 변경하고 p의 다음 노드를 첫 번째 노드로 변경하며 두 개의 노드의 위치를 서로 뒤바꾼다. 현재 p는 첫 번째 노드를 가리키고 있으며, prev가 첫 노드를 다시 가리키게 만들어준 뒤, p를 p의 다음 노드로 대입해 2번째 노드를 가리키게 해준다. 그 후 첫 번째 노드를 fir에 저장한다.

```
83              else
84              {
85                  if (p->data.stu_num > p->link->data.stu_num)
86                  {
87                      prev->link = p->link;//p노드의 전 노드를 p노드의 다음노드 바로 전에 오도록 연결
88                      p->link = p->link->link;//p노드가 p->link 노드에 위치해야 하기때문에 p노드를 p->link->link 바로 전에 오도록 연걸
89                      prev->link->link = p;//원래 p의 다음노드였던 노드를 p의 바로 전에 연결
90                      prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞으로 이동
91                  }
92                  else
93                  {
94                      p = p->link;//p다음 노드가 p보다 큰 경우이기 때문에 비교할 데이터를 p->link로 대입
95                      prev = prev->link;//p가 한칸 이동 했기 때문에 prev 노드도 한칸 앞으로 이동
96                  }
97              }
98          }
99      }
```

두 번째 노드에서 부터는 p와 그다음 노드의 값을 비교한다. 만약 p의 값이 다음 노드의 데이터보다 크다면 p의 전 노드 (prev)의 다음 노드를 p의 다음 노드 (p->link)로 만들어 p 노드에 뒤에 놓고 p의 다음 노드를 p의 다다음 노드로 만들어 p 노드를 비교를 진행한 p->link에 앞질러 놓고 prev의 다다음 노드 (비교 전 p->link가 가리키는 노드)를 p로 만들어 p와 연결시켜 일렬로 연결시켜준다. 그 후 prev 포인터를 한 칸 다음으로 이동시킨다. 만약 p의 데이터보다 p의 다음 노드의 데이터가 크다면 p 포인터가 더 큰 값을 가진 (p->link)로 갱신 시키고 prev도 마찬가지로 따라 한 칸 앞으로 옮겨준다. 이러한 반복을 통해 정렬을 마친 뒤 첫 번째 노드를 저장한 fir 포인터를 리턴한다.

```
218 // 테스트 프로그램
219 int main(void)
220 {
221     ListNode* head = NULL;
222
223     FILE* fp;
224     fp=fopen("data.txt", "r");
225     while (!feof(fp))
226     {
227         element data;
228         char name[100];
229         fscanf(fp,"%d %s %d", &data.stu_num, name, &data.score);
230         data.name = (char*)malloc(strlen(name) + 1);
231         strcpy(data.name, name);
232         //연결리스트에 추가
233         head=insert_first(head,data);
234     }
235
236     //node개수 탐색
237     int Count = Count_Node(head);
238     //학번순 정렬 후 출력
239     head = Sort_Stunum(head, Count);
240     printf("============================================================WnWtWt학번순 정렬 (오름차순)Wn============================================================Wn");
241     print_list(head);
242
243     //이름순 정렬 후 출력
244     head = Sort_Name(head, Count);
```

```
245     printf("₩n=============================================================₩n₩t₩t이름순 ↵
           정렬 (ㄱ ~ ㅎ)₩n=============================================================₩n ↵
          ₩n");
246     print_list(head);
247
248
249     //총점순 정렬 후 출력
250     head = Sort_Score(head, Count);
251     printf("₩n=============================================================₩n₩t₩t총점순 ↵
           정렬 (내림차순)₩n=============================================================₩n↵
          ₩n");
252     print_list(head);
253
254     //동적할당 메모리 해제
255     node_free(head);
256
257     return 0;
258 }
```

main 함수에 경우 파일을 불러 data를 저장한 후 이름의 크기에 맞도록 동적할당을 해 저장해 메모리를 효율적으로 사용하며 노드를 추가시킨다. 파일이 끝날 때까지 반복한다. 각 정렬 함수를 호출해 정렬 후 출력한다. 그 후 동적할당받은 메모리를 해제시켜준다.

```
205 //동적할당 해제 함수
206 void node_free(ListNode* head)
207 {
208     while (head!=NULL)
209     {
210         ListNode* remove=head;
211         head = head->link;
212         free(remove->data.name);
213         free(remove);
214     }
215 }
```

메모리 해제는 이름 저장을 위해 받은 메모리 먼저 해제하고 그다음 동적할당받은 노드를 해제한다. head를 remove에 저장하기 때문에 첫 노드부터 차례대로 해제된다.

5. 실행창

```
==========================================================
                학번순 정렬 (오름차순)
==========================================================

학번            이름                    성적

20120121        홍길동                  196
20170234        화이자                  167
20190171        얀센                    157
20200012        이순신                  173
20200151        아스트라제네카          177
20210011        김욱                    176
20229812        아리스토텔레스          182
20230321        강감찬                  183
20240013        김유신                  183

==========================================================
                이름순 정렬 (ㄱ ~ ㅎ)
==========================================================

학번            이름                    성적

20230321        강감찬                  183
20210011        김욱                    176
20240013        김유신                  183
20229812        아리스토텔레스          182
20200151        아스트라제네카          177
20190171        얀센                    157
20200012        이순신                  173
20120121        홍길동                  196
20170234        화이자                  167

==========================================================
                총점순 정렬 (내림차순)
==========================================================

학번            이름                    성적

20120121        홍길동                  196
20230321        강감찬                  183
20240013        김유신                  183
20229812        아리스토텔레스          182
20200151        아스트라제네카          177
20210011        김욱                    176
20200012        이순신                  173
20170234        화이자                  167
20190171        얀센                    157
```

6. 느낀 점

버블 정렬의 효율성은 정렬 알고리즘 중 가장 좋지 않은 방법으로 알고 있었다. 이론적인 것 을 넘어 직접 코드를 작성하며 그 이유를 탐색할 수 있던 경험이었다. 또한 데이터만을 정렬시키는 것이 아닌 노드 자체를 옮기다 보니 연결 리스트, 더 나아가 포인터에 대한 개념까지 다시 한번 살펴볼 수 있었던 기회이기도 했다.