

# REPORT

## 전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: <http://www.ieee.org>)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임:** 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제:** 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성:** 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지:** 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해:** 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성:** 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세:** 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기:** 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.
9. **도덕성:** 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애:** 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

과제2. 포인터와 함수활용

학 부: 전자공학과

제출일: 2022. 04. 13

과목명: 전자공학프로그래밍

교수명: 이 미 언 교수님

분 반: C038-2

학 번: 201921021

성 명: 석 대 근

## 1) 소스코드 분석

### 1. main 함수 이전

```
1 //과제02. 전자공학과 201921021 석대근
2
3 #define _CRT_SECURE_NO_WARNINGS //scanf, gets 함수를 사용하기 위해 사용
4 #include <stdio.h>
5 #include <string.h> //strcpy, strcmp, strcmp 함수, memset 함수를 사용하게 해주는 라이브러리
6 #include <stdlib.h> // atoi 함수(문자열을 정수형숫자로 변경해주는 함수)를 사용하는 라이브러리
7
8 #define MAX_COLUMN 100 //범례의 최대 저장 길이를 define 상수로 표현(값 100)
9 #define MAX_DATA_LENGTH 100 //데이터의 최대 저장 길이를 define 상수로 표현(값 100)
10 #define MAX_DATA_NUM 100 //데이터 최대 저장 개수를 define 상수로 표현(값 100)
11
12 typedef struct {
13     char column[MAX_COLUMN]; //범례 정보를 저장해주는 MAX_COLUMN칸의 column 배열 선언
14     char data[MAX_DATA_NUM][MAX_DATA_LENGTH]; //데이터 정보를 저장해주는 MAX_DATA_NUM*MAX_DATA_LENGTH칸의 data 배열 선언
15 }Player; //선수들의 정보를 갖고 있는 Player 형을 typedef를 통해 선언
16
17 typedef struct {
18     char column[MAX_COLUMN];
19     char data[MAX_DATA_NUM][MAX_DATA_LENGTH];
20 }Club; //팀의 정보를 갖고 있는 Club 형을 typedef를 통해 선언(내용은 Player형의 설명과 동일한 내용)
21
22 typedef struct {
23     char column[MAX_COLUMN];
24     char data[MAX_DATA_NUM][MAX_DATA_LENGTH];
25 }League; //경기 내용의 정보를 갖고 있는 League 형을 typedef를 통해 선언(내용은 Player형의 설명과 동일한 내용)
26
27 void set_table(Player* table_Player, Club* table_Club, League* table_League); //구조체에 저장된 데이터를 txt파일로 저장하여 주는 함수인 set_table 함수 선언
28 void get_all_element(Player* table_Player, Club* table_Club, League* table_League); //txt 파일에 저장된 정보를 구조체에 삽입해주는 함수인 get_all_element 함수 선언
29 void print_all_table(Player* table_Player, Club* table_Club, League* table_League); //현재 저장되어 있는 구조체의 데이터를 전부 출력해주는 함수인 print_all_table 함수 선언
30 void find_element_by_key(Player* table_Player, Club* table_Club, League* table_League); //데이터의 key값을 이용해서 다른 데이터를 찾는 함수 find_element_by_key 함수 선언
31
32 //추가구현한 함수로, data[]에 저장되어 있는 값을 ','로 나눠주기 위해 구현한 함수들이다.
33 void split_p(Player* table_Player, char* str, int key, char* sarr[MAX_DATA_NUM]);
34 void split_c(Club* table_Club, char* str, int key, char* sarr[MAX_DATA_NUM]);
35 void split_l(League* table_League, char* str, int key, char* sarr[MAX_DATA_NUM]);
36
37
38
39
40
```

### 2. main 함수

```
42 int main(void) {
43     //초기 설정 단계: Player형 구조체 table_Player의 column[MAX_COLUMN], data[MAX_DATA_NUM][MAX_DATA_LENGTH]에 모두 0을 저장하게 한다. table_Club, table_League 역시 형만 Club형, League형일 뿐 실제 설명과 동일하게 진행하였다
44     Player table_Player = { (0),(0) };
45     Club table_Club = { (0),(0) };
46     League table_League = { (0),(0) };
47
48     Player* Table_p = &table_Player; //table_Player 구조체의 시작 주소를 가리키는 포인터 변수 Table_p 선언
49     Club* Table_c = &table_Club; //table_Club 구조체의 시작 주소를 가리키는 포인터 변수 Table_c 선언
50     League* Table_l = &table_League; //table_League 구조체의 시작 주소를 가리키는 포인터 변수 Table_l 선언
51
52     /* 실행할 때 글러서 사용하면 됨
53     set_table(Table_p, Table_c, Table_l);
54     get_all_element(Table_p, Table_c, Table_l);
55     print_all_table(Table_p, Table_c, Table_l);
56     find_element_by_key(Table_p, Table_c, Table_l);
57     */
58
59     return 0; //main 함수가 정상적으로 종료되었다는 것을 알리기 위해서 사용되었다.
60 }
61
62
```

### 3. 함수 정의 부분

#### 1) 구조체에 저장된 데이터를 txt파일로 저장하여 주는 함수: set\_table 함수

```
64 //구조체에 저장된 데이터를 txt파일로 저장하여 주는 함수인 set_table 함수의 정의 부분
65 void set_table(Player* table_Player, Club* table_Club, League* table_League) { //함수의 인자로는 Player형 주소를 갖는 포인터 변수, Club형 주소를 갖는 포인터 변수, League형 주소를 갖는 포인터 변수를 사용했고
66     //Call By Reference 방식을 사용하였다.
67     int fp_inum = 0; //
68     FILE* fp1 = NULL, * fp2 = NULL, * fp3 = NULL; //파일형 포인터 변수 fp1,fp2,fp3를 선언하고 아무것도 가리키지 않는 NULL을 가리키도록 하였다.
69
70     //왼쪽 형의 column 배열에 오른쪽 문자열을 넣어주기 위해 strcpy 함수를 사용해주었다.
71     strcpy(table_Player->column, "key,p_name,club_key\n");
72     strcpy(table_Club->column, "key,c_name,League_key\n");
73     strcpy(table_League->column, "key,club_key1,club_key2,score\n");
74
75     //왼쪽 형의 data 배열에 오른쪽 문자열을 넣어주기 위해 strcpy를 사용해주었다.(table_Player 구조체의 data 배열의 각 행 부분에 아래의 정보들이 들어가게 된다.)
76     strcpy(table_Player->data[0], "1,손흥민,1\n");
77     strcpy(table_Player->data[1], "2,만유,1\n");
78     strcpy(table_Player->data[2], "3,박주영,2\n");
79
80     //왼쪽 형의 data 배열에 오른쪽 문자열을 넣어주기 위해 strcpy를 사용해주었다.(table_Club 구조체의 data 배열의 각 행 부분에 아래의 정보들이 들어가게 된다.)
81     strcpy(table_Club->data[0], "1,토트넘,1\n");
82     strcpy(table_Club->data[1], "2,만유,1\n");
83     strcpy(table_Club->data[2], "3,아스날,2\n");
84     strcpy(table_Club->data[3], "4,맨시티,2\n");
85
86     //왼쪽 형의 data 배열에 오른쪽 문자열을 넣어주기 위해 strcpy를 사용해주었다.(table_League 구조체의 data 배열의 각 행 부분에 아래의 정보들이 들어가게 된다.)
87     strcpy(table_League->data[0], "1,1,2,1:2\n");
88     strcpy(table_League->data[1], "2,3,4,2:3\n");
89
90
91     fp1 = fopen("Player.txt", "w"); //fopen 함수를 사용하여 "Player.txt" 파일을 쓰기 모드로 생성한다. 추가(append) 모드가 아니기 때문에 다시 실행할 경우 이전 내용이 지워지고 파일의 처음부터 쓰기 시작한다.
92     fp2 = fopen("Club.txt", "w"); //fopen 함수를 사용하여 "Club.txt" 파일을 쓰기 모드로 생성한다. 추가(append) 모드가 아니기 때문에 다시 실행할 경우 이전 내용이 지워지고 파일의 처음부터 쓰기 시작한다.
93     fp3 = fopen("League.txt", "w"); //fopen 함수를 사용하여 "League.txt" 파일을 쓰기 모드로 생성한다. 추가(append) 모드가 아니기 때문에 다시 실행할 경우 이전 내용이 지워지고 파일의 처음부터 쓰기 시작한다.
94
95     //fputs 함수를 통하여 각 구조체 안의 column 배열에 저장된 문자열 내용을 txt파일에 기록해주는 구문이다.
96     fputs(table_Player->column, fp1);
97     fputs(table_Club->column, fp2);
98     fputs(table_League->column, fp3);
99
```

```

100  /* fputs 함수를 통하여 각 구조체 안의 data 배열에 저장된 문자열 내용을 txt파일에 기록해주는 구문이다.
101  data 배열은 column 배열과 다른 2차원 배열이고, 행 부분에 문자열 내용을 저장하므로 for문을 사용하여 data[fp_innum]의 문자열 내용들을 차례대로 txt파일에 기록할 수 있도록 하였다. */
102  for (fp_innum = 0; fp_innum < MAX_DATA_NUM; fp_innum++) //원래는 3이었으나, 해당 코드의 적용 범위를 더 넓게 해주기 위하여 데이터 최대 저장 개수인 MAX_DATA_NUM을 사용해주었다
103      fputs(table_Player->data[fp_innum], fp1);
104
105  for (fp_innum = 0; fp_innum < MAX_DATA_NUM; fp_innum++) //원래는 4이었으나, 해당 코드의 적용 범위를 더 넓게 해주기 위하여 데이터 최대 저장 개수인 MAX_DATA_NUM을 사용해주었다
106      fputs(table_Club->data[fp_innum], fp2);
107
108  for (fp_innum = 0; fp_innum < MAX_DATA_NUM; fp_innum++) //원래는 2이었으나, 해당 코드의 적용 범위를 더 넓게 해주기 위하여 데이터 최대 저장 개수인 MAX_DATA_NUM을 사용해주었다
109      fputs(table_League->data[fp_innum], fp3);
110
111  //파일 입출력 수행 후에는 반드시 파일을 닫아주어야 하며, 그렇게 하지 않을 경우 메모리에 계속 남아있게 된다. 때문에 fclose함수를 사용하여 파일을 닫아주었다.
112  fclose(fp1);
113  fclose(fp2);
114  fclose(fp3);
115  }
116

```

set\_table 함수에서는 각 구조체 변수의 범례, 데이터에 코드 내에서 지정된 값들을 대입해주고, txt 파일을 생성하여 저장해주는 함수이다. 범례를 저장하는 column의 경우 행이 하나이므로 반복문 없이 값을 대입해줘도 되지만, data의 경우 MAX\_DATA\_NUM개만큼 행을 구성할 수 있으므로 반복문인 for문을 통하여 값들을 대입해주었다.

## 2) txt파일에 저장된 정보를 구조체에 삽입해주는 함수: get\_all\_element 함수

```

117  //txt파일에 저장된 정보를 구조체에 삽입해주는 함수인 get_all_element 함수의 정의 부분
118  void get_all_element(Player* table_Player, Club* table_Club, League* table_League) { //함수의 인자로 Player형 주소를 갖는 포인터 변수, Club형 주소를 갖는 포인터 변수, League형 주소를 갖는 포인터 변수를 사용했고
119  //Call By Reference 방식을 사용하였다.
120  //memset 함수를 통해 각 구조체에 담겨있는 정보들을 초기화해주었다.
121  memset(table_Player, 0, sizeof(Player));
122  memset(table_Club, 0, sizeof(Club));
123  memset(table_League, 0, sizeof(League));
124
125  //파일할 포인터 변수 fp1,fp2,fp3를 선언하고 아무것도 가리키지 않는 NULL를 가리키도록 하였다.
126  FILE* fp1 = NULL, * fp2 = NULL, * fp3 = NULL;
127
128
129  fp1 = fopen("Player.txt", "r"); //fopen 함수를 사용하여 "Player.txt" 파일을 읽기 모드로 생성한다. 쓰기/추가 모드가 아니기 때문에 해당 txt파일에 기록하는 것은 불가능하게 하였다.
130  fp2 = fopen("Club.txt", "r"); //fopen 함수를 사용하여 "Club.txt" 파일을 읽기 모드로 생성한다. 쓰기/추가 모드가 아니기 때문에 해당 txt파일에 기록하는 것은 불가능하게 하였다.
131  fp3 = fopen("League.txt", "r"); //fopen 함수를 사용하여 "League.txt" 파일을 읽기 모드로 생성한다. 쓰기/추가 모드가 아니기 때문에 해당 txt파일에 기록하는 것은 불가능하게 하였다.
132
133  int p = 0, c = 0, l = 0; //Table_Player, Table_Club, Table_League의 data 배열에 들어가는 개수를 파악하기 위해 만든 변수 p,c,l을 0으로 둘
134  fgetc(table_Player->column, MAX_COLUMN, fp1); //fgetc를 사용하여 Table_Player의 column 배열에 들어가는 내용을 읽어온다
135  while (fgetc(table_Player->data[p], MAX_DATA_LENGTH, fp1)) //fgetc를 사용하여 Table_Player의 data[p]에 들어가는 내용을 읽어온다. 만약 data[p]에 내용이 없다면, while문을 종료한다
136      p++; //p를 1씩 증가시켜준다
137
138  fgetc(table_Club->column, MAX_COLUMN, fp2); //fgetc를 사용하여 Table_Club의 column 배열에 들어가는 내용을 읽어온다
139  while (fgetc(table_Club->data[c], MAX_DATA_LENGTH, fp2)) //fgetc를 사용하여 Table_Club의 data[c]에 들어가는 내용을 읽어온다. 만약 data[c]에 내용이 없다면, while문을 종료한다
140      c++; //c를 1씩 증가시켜준다
141
142  fgetc(table_League->column, MAX_COLUMN, fp3); //fgetc를 사용하여 Table_League의 column 배열에 들어가는 내용을 읽어온다
143  while (fgetc(table_League->data[l], MAX_DATA_LENGTH, fp3)) //fgetc를 사용하여 Table_League의 data[l]에 들어가는 내용을 읽어온다. 만약 data[l]에 내용이 없다면, while문을 종료한다
144      l++; //l를 1씩 증가시켜준다
145
146  //파일 입출력 수행 후에는 반드시 파일을 닫아주어야 하며, 그렇게 하지 않을 경우 메모리에 계속 남아있게 된다. 때문에 fclose함수를 사용하여 파일을 닫아주었다.
147  fclose(fp1);
148  fclose(fp2);
149  fclose(fp3);
150  }
151

```

각 구조체형 변수 table\_Player, table\_Club, table\_League에 들어있는 column(범례) 정보, data(내용) 정보를 모두 초기화하기 위해 배열이나 구조체 등의 메모리의 모든 내용을 0으로 설정할 때 많이 사용하는 함수 memset 함수를 사용하였다. 그다음, 각 txt 파일들을 읽기 모드로 열어주고 '\n'을 만나기 전까지 공백을 포함한 모든 입력을 읽어주는 fgetc 함수를 통해 txt파일에 적혀 있는 범례 정보, 데이터 정보들을 읽어오도록 코드를 설계하였다. 범례 정보는 한 개이므로 반복문을 사용할 필요가 없지만, 데이터 정보들은 MAX\_DATA\_NUM개가 있기에 반복문인 while문을 사용하여 fgetc함수가 읽어오는 것이 없을 때까지 반복되도록 설계하였다.



### 3) 현재 저장되어 있는 구조체의 데이터를 전부 출력해주는 함수: print\_all\_table 함수

```
152 //현재 저장되어 있는 구조체의 데이터를 전부 출력해주는 함수인 print_all_table 함수의 정의 부분
153 void print_all_table(Player* table_Player, Club* table_Club, League* table_League) { //함수의 인자로는 Player형 주소를 갖는 포인터 변수, Club형 주소를 갖는 포인터 변수, League형 주소를 갖는 포인터 변수를 사용했고
154 //Call By Reference 방식을 사용하였다.
155 int p, c, l; //table_Player, table_Club, table_League의 data 배열에 들어있는 개수를 파악하기 위해 만든 변수 p, c, l 선언
156 printf("-----Player table-----\n");
157 for (p = 0; table_Player != 0; p++) //table_Player 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 p를 1 증가시켜준다
158 {
159     if (p == 0)
160         printf("%s", table_Player->column); //p=0이면 column 배열의 내용을 출력하도록 함
161     else {
162         if (table_Player->data[p - 1][0] == 0) //data[p-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
163             break;
164         printf("%s", table_Player->data[p - 1]); //data[p-1]의 내용을 출력하도록 함
165     }
166 }
167
168 printf("-----Club table-----\n");
169 for (c = 0; table_Club != 0; c++) //table_Club 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 c를 1 증가시켜준다
170 {
171     if (c == 0)
172         printf("%s", table_Club->column); //c=0이면 column 배열의 내용을 출력하도록 함
173     else {
174         if (table_Club->data[c - 1][0] == 0) //data[c-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
175             break;
176         printf("%s", table_Club->data[c - 1]); //data[c-1]의 내용을 출력하도록 함
177     }
178 }
179
180 printf("-----League table-----\n");
181 for (l = 0; table_League != 0; l++) //table_League 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 l를 1 증가시켜준다
182 {
183     if (l == 0)
184         printf("%s", table_League->column); //l=0이면 column 배열의 내용을 출력하도록 함
185     else {
186         if (table_League->data[l - 1][0] == 0) //data[l-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
187             break;
188         printf("%s", table_League->data[l - 1]); //data[l-1]의 내용을 출력하도록 함
189     }
190 }
191 }
```

print\_all\_table 함수는 현재 구조체 변수 table\_Player, table\_Club, table\_League에 저장되어있는 데이터들을 모두 출력해주는 함수이다. data의 행 index를 가리키는 변수 p, c, l을 통하여 데이터가 적혀있는 data 배열의 내용들을 출력해주고자 하였다. 우선, 각 행 index를 가리키는 변수가 0일 때 범례가 적혀있는 column 배열의 내용을 먼저 출력해준 뒤, 데이터가 적혀있는 data 배열의 내용들을 index-1을 통하여 출력해주었고, 반복문 안에 있는 명령들을 다 수행하고 나면 index의 값을 1씩 증가시켜주었다. 그러던 중 비어있는 값을 만나면 break를 통하여 반복문을 빠져나가도록 설계하였다.

### 4) 데이터의 key값을 이용해서 다른 데이터를 찾는 함수: find\_element\_by\_key 함수

```
193 //데이터의 key값을 이용해서 다른 데이터를 찾는 함수 find_element_by_key 함수의 정의 부분
194 void find_element_by_key(Player* table_Player, Club* table_Club, League* table_League) { //함수의 인자로는 Player형 주소를 갖는 포인터 변수, Club형 주소를 갖는 포인터 변수, League형 주소를 갖는 포인터 변수를 사용했고
195 //Call By Reference 방식을 사용하였다.
196 char select_mode_table[100] = { '0' }; //select_mode_table이라는 char형 배열(=문자열)을 선언하고 어떤 테이블을 사용할지 받아주는데 사용된다.
197 char start_sign[5] = { '0' }; //처음 key를 입력할 때 사용되는 char형 배열(=문자열) start_sign을 선언해주고, 초기값을 '0'으로 채워준다.
198 char sarr[100] = { NULL }; //split에서 사용될 char형 포인터 배열 sarr[100]을 선언해주고, 초기값을 NULL로 채워준다.
199 int key = 0; //처음 입력받은 key를 담고 있는 int형 변수 key이고 초기값은 0이다.
200 int next_key = 0; //다음 table에 있는 정보를 가리키는 key를 담고 있는 int형 변수 next_key이고 초기값은 0이다.
201 char str[MAX_DATA_LENGTH] = { '0' }; //char형 배열 str의 길이를 data 배열의 최대 데이터 길이, MAX_DATA_LENGTH와 동일하게 설정하여 문자를 받을 때 오류가 발생하지 않도록 설정하였다.
202
203 printf("어떤테이블을 사용하시겠습니까? (table_Player, table_Club, table_League) : ");
204 gets(select_mode_table); //gets 함수를 사용하여 'n'을 입력받기 전까지의 내용을 select_mode_table 문자열에 저장해준다.
205 printf("-----데이터를 구성내용-----\n");
206
207 if (strcmp(select_mode_table, "table_Player") == 0) { //strcmp함수를 사용하여 두 문자열이 같은 문자열일 경우 0이 출력되므로 0==0이 성립하게 되어 if문이 실행되도록 하였다.
208     for (int i = 0; table_Player != 0; i++) //table_Player 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 i를 1 증가시켜준다
209     {
210         if (i == 0) //i=0이면 column 배열의 내용을 출력하도록 함
211             printf("%s", table_Player->column);
212         else {
213             if (table_Player->data[i - 1][0] == 0) //data[i-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
214                 break;
215             printf("%s", table_Player->data[i - 1]); //data[i-1]의 내용을 출력하도록 함
216         }
217     }
218     while (1) { //실행 실패 예에서 반복시켜주는 것을 확인하기 위해 key값이 0일 경우에만 해당 함수가 종료되도록 반복문을 설정하였다
219         printf("n");
220         printf("연관된 자료를 찾을 key를 입력하십시오(>종료) : ");
221         gets(start_sign); //gets 함수를 사용하여 'n'을 입력받기 전까지의 내용을 start_sign에 보관해준다. start_sign은 숫자 하나(char형)를 입력받아야 함을 유의하자.
222         key = atoi(start_sign); //char to int = 문자열을 정수 타입으로 바꿔주는 atoi함수를 통하여 문자로 보관하고 있던 숫자값을 정수 숫자로 바꿔준다 그 다음 int형 변수 key에 그 값을 저장해준다.
223         if (key == 0) return; //key값이 0일 경우 해당 함수가 종료되어야 하므로 return을 통해 find_element_by_key 함수를 즉시 종료시켜주었다.
224         printf("데이터를 검색할 %s", table_Player->column);
225         printf("검색된 데이터: %s", table_Player->data[key - 1]); //table_Player 구조체의 data 배열에 있는 key별 문자열을 출력해주고, 이것이 우리가 찾고 있는 data이다.
226         split_p(table_Player, str, key, sarr); //추가구현한 함수: split_p를 실행해준다. 함수의 parameter로는 table_Player, str, key, sarr이 사용되었다. split_p 함수를 실행하고 나면 sarr[2]에 club_key를 나타내는 값이 저장되어 있게 된다.
227         next_key = atoi(sarr[2]); //문자열을 정수형으로 변환해주는 atoi함수를 사용해 club_key 값이 next_key라는 int형 변수에 정수로 저장되도록 하였다.
228         printf("키값으로 찾은 클럽정보: %s", table_Club->data[next_key - 1]); //다음 next_key값을 통해 table_Player에 연관되어 있는 다른 table: table_Club의 next_key에 대응하는 data를 출력한다.
229     }
230 }
231
232
233 if (strcmp(select_mode_table, "table_Club") == 0) { //strcmp함수를 사용하여 두 문자열이 같은 문자열일 경우 0이 출력되므로 0==0이 성립하게 되어 if문이 실행되도록 하였다.
234     for (int i = 0; table_Club != 0; i++) //table_Club 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 i를 1 증가시켜준다
235     {
236         if (i == 0) //i=0이면 column 배열의 내용을 출력하도록 함
237             printf("%s", table_Club->column);
238         else {
239             if (table_Club->data[i - 1][0] == 0) //data[i-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
240                 break;
241             printf("%s", table_Club->data[i - 1]); //data[i-1]의 내용을 출력하도록 함
242         }
243     }
244     while (1) { //실행 실패 예에서 반복시켜주는 것을 확인하기 위해 key값이 0일 경우에만 해당 함수가 종료되도록 반복문을 설정하였다
245         printf("n");
246         printf("연관된 자료를 찾을 key를 입력하십시오(>종료) : ");
247         gets(start_sign); //gets 함수를 사용하여 'n'을 입력받기 전까지의 내용을 start_sign에 보관해준다. start_sign은 숫자 하나(char형)를 입력받아야 함을 유의하자.
248         key = atoi(start_sign); //char to int = 문자열을 정수 타입으로 바꿔주는 atoi함수를 통하여 문자로 보관하고 있던 숫자값을 정수 숫자로 바꿔준다 그 다음 int형 변수 key에 그 값을 저장해준다.
249         if (key == 0) return; //key값이 0일 경우 해당 함수가 종료되어야 하므로 return을 통해 find_element_by_key 함수를 즉시 종료시켜주었다.
250         printf("데이터를 검색할 %s", table_Club->column);
251         printf("검색된 데이터: %s", table_Club->data[key - 1]); //table_Club 구조체의 data 배열에 있는 key별 문자열을 출력해주고, 이것이 우리가 찾고 있는 data이다.
252         split_c(table_Club, str, key, sarr); //추가구현한 함수: split_c를 실행해준다. 함수의 parameter로는 table_Club, str, key, sarr이 사용되었다. split_c 함수를 실행하고 나면 sarr[2]에 League_key를 나타내는 값이 저장되어 있게 된다.
253         next_key = atoi(sarr[2]); //문자열을 정수형으로 변환해주는 atoi함수를 사용해 League_key 값이 next_key라는 int형 변수에 정수로 저장되도록 하였다.
254         printf("키값으로 찾은 경기정보: %s", table_League->data[next_key - 1]); //다음 next_key값을 통해 table_Club에 연관되어 있는 다른 table: table_League의 next_key에 대응하는 data를 출력한다.
255     }
256 }
```

```

262 if (strcmp(select_mode_table, "table_League") == 0) { //strcmp함수를 사용하여 두 문자열이 같은 문자열일 경우 0이 출력되므로 0==0이 성립하게 되어 if문이 실행되도록 하였다.
263     for (int i = 0; table_League != 0; i++) //table_League 포인터가 가리키는 것이 NULL이 아닐 때, 이 반복문을 수행하게 하고, 해당 루프가 끝나면 i를 1 증가시켜준다.
264     {
265         if (i == 0) //i==0이면 column 배열의 내용을 출력하도록 함
266             printf("%s", table_League->column);
267         else {
268             if (table_League->data[i - 1][0] == 0) //data[i-1][0]이 NULL일 경우 해당 반복문을 빠져나가도록 함
269                 break;
270             printf("%s", table_League->data[i - 1]); //data[i-1]의 내용을 출력하도록 함
271         }
272     }
273     while (1) { //실제 실행 예시에서 반복시켜주는 것을 확인했기에 key값이 0일 경우에만 해당 함수가 종료되도록 반복문을 설정하였다
274         printf("\n");
275         printf("연관된 자료를 찾을 key를 입력하십시오(0=종료) : ");
276         gets(start_sign); //gets 함수를 사용하여 '\n'을 입력받기 전까지의 내용을 start_sign에 보관해준다. start_sign은 숫자 하나(char형)를 입력받아야 할을 유의하자.
277
278         key = atoi(start_sign); //char to int = 문자열을 정수 타입으로 바꿔주는 atoi함수를 통하여 문자로 보관하고 있던 숫자값을 정수 숫자로 바꿔준다 그 다음 int형 변수 key에 그 값을 저장해준다.
279         if (key == 0) return; //key가 0일 경우 해당 함수가 종료되어야 하므로 return을 통해 find_element_by_key 함수를 즉시 종료시켜주었다.
280         printf("키값으로 찾은 클럽정보: %s", table_Club->data[next_key - 1]); //얻은 next_key값을 통해 table_League에 연관되어 있는 다른 table: table_Club의 next_key에 대응하는 data를 출력한다.
281         printf("선택된 데이터: %s", table_League->data[key - 1]);
282
283         split_1((table_League, str, key, sArr); //추가구현한 함수: split_1을 실행해준다. 함수의 parameter로는 table_League, str, key, sArr이 사용되었다.
284         //split_1 함수를 실행하고 나면 sArr[1]에 club_key1, sArr[2]에 club_key2를 나타내는 값이 저장되어 있게 된다.(앞의 두 개의 차이점[주의])
285         next_key = atoi(sArr[1]); //문자열을 정수형으로 변환해주는 atoi함수를 사용해 club_key1 값이 next_key라는 int형 변수에 정수로 저장되도록 하였다.
286         printf("키값으로 찾은 클럽정보: %s", table_Club->data[next_key - 1]); //얻은 next_key값을 통해 table_League에 연관되어 있는 다른 table: table_Club의 next_key에 대응하는 data를 출력한다.
287         next_key = atoi(sArr[2]); //문자열을 정수형으로 변환해주는 atoi함수를 사용해 club_key2 값이 next_key라는 int형 변수에 정수로 저장되도록 하였다.
288
289         printf("키값으로 찾은 클럽정보: %s", table_Club->data[next_key - 1]); //얻은 next_key값을 통해 table_League에 연관되어 있는 다른 table: table_Club의 next_key에 대응하는 data를 출력한다.
290     }
291 }
292 }

```

우선 어떤 table에 접근할 것인지 입력받기 위해 select\_mode\_table라는 char형 배열(문자열)을 사용하였고, ‘\n’을 만나기 전까지 공백을 포함한 모든 입력을 읽어주는 gets함수를 통해 select\_mode\_table에 넣어준다. 그다음 어떤 table을 사용하게 되는지를 strcmp를 통해 확인해주고, 일치하는 table의 범례, 데이터를 출력해준다. 그 뒤, key를 입력받는데 여기서 입력받는 key는 문자형이기에 문자열을 정수 타입으로 바꿔주는 atoi 함수를 통해 정수형의 key값을 얻게 되고 이를 통해 해당 table의 데이터를 출력할 수 있게 된다.

문제는 데이터 안에 있는 다른 table의 데이터를 어떻게 출력해주자인데, 이를 split\_p, split\_c, split\_l 함수를 새로 구현하여 sArr이라는 포인터 배열에 각각 다른 table의 key 값을 가리키도록 나오도록 설계했고, 다른 table의 데이터를 출력할 수 있었다.

table\_League의 경우 table\_Player, table\_Club과 범례가 다른데 이를 주의해서 함수를 설계해야 하며 다른 table의 key값이 2개 들어있으므로 next\_key에 key값을 얻은 뒤, printf 함수를 통해 바로 출력해주고 다시 next\_key에 key값을 얻어주는 방식으로 함수를 설계하였다.

## 5) 추가구현한 함수: data[]에 저장되어 있는 값을 ','로 나눠 다른 데이터의 key들을 얻기 위해 구현한 함수

```

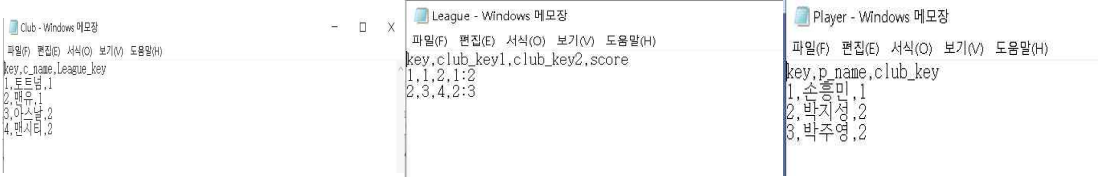
294 //table_Player의 data 형에는 key_p.name, club_key의 정보가 저장되어 있다. 다음 데이터의 key(next_key)를 따로 저장하는 것이 중요함데
295 //추가구현한 split_1함수로 key값을 통해 data[key - 1]에 저장된 club_key의 값을 sArr[2]에 저장하도록 함수를 설계 하였다. ~/
296
297 void split_p(Player* table_Player, char* str, int key, char* sArr[MAX_DATA_NUM]) {
298     strcpy(str, table_Player->data[key - 1]); //원본 앞의 str 배열에 key를 통해 얻어낸 data[key-1]의 내용을 넣어주기 위해 strcpy를 사용해주었다.
299     char* ptr = strtok(str, ","); //구분자(여기서는 ,)를 기준으로 문자열을 잘라서 문자열의 포인터를 반환하는 함수 strtok를 사용하여 char형 포인터 변수 ptr에 넣어주었다. 여기서 ptr은 첫 번째 ,를 만나기 전까지의 내용인 key의 포인터를 저장하게 된다.
300
301     int i = 0; //char형 포인터 배열 sArr에 char형 포인터 ptr를 순서대로 넣어주기 위한 index i를 0으로 설정한다
302     while (ptr != NULL) //ptr이 NULL이 아닐 때 해당 반복문을 수행하도록 한다. ptr이 NULL인 경우, str에 있는 모든 문자가 읽힌 것이고 이 경우 반복문을 종료하도록 하였다.
303     {
304         sArr[i] = ptr; //포인터 배열 sArr[i]에 나뉜 ptr의 포인터를 저장하게 된다. sArr[0]은 key의 값을 저장하고, sArr[1]은 p.name의 값을 저장하고, sArr[2]는 club_key의 값을 저장하게 된다.
305         ++i; //sArr의 index, i를 1 증가시켜준다
306         ptr = strtok(NULL, ","); //첫 번째 만나준 다음부터는 strtok를 다음과 같이 사용해야 한다.
307     }
308     return;
309 }
310
311
312 //split_c 함수는 split_p 함수와 완전히 동일하게 함수가 설계되었다. 하지만, 함수를 다르게 지정해 준 이유는 typedef를 통해 형을 다르게 설정해주었기 때문에 함수에서 parameter로 넘겨줘야 하는 인자가 다르므로 함수를 다르게 정의하였다.
313 void split_c(Club* table_Club, char* str, int key, char* sArr[MAX_DATA_NUM]) {
314     strcpy(str, table_Club->data[key - 1]);
315     char* ptr = strtok(str, ",");
316     int i = 0;
317     while (ptr != NULL)
318     {
319         sArr[i] = ptr; //포인터 배열 sArr[i]에 나뉜 ptr의 포인터를 저장하게 된다. sArr[0]은 key의 값을 저장하고, sArr[1]은 c.name의 값을, sArr[2]는 League_key의 값을 저장하게 된다.
320         ++i;
321         ptr = strtok(NULL, ",");
322     }
323     return;
324 }
325
326
327 //split_l 함수 역시 split_p 함수와 완전히 동일하게 설계되었다.하지만, 함수를 다르게 지정해 준 이유는 typedef를 통해 형을 다르게 설정해주었기 때문에 함수에서 parameter로 넘겨줘야 하는 인자가 다르므로 함수를 다르게 정의하였다.
328 void split_l(League* table_League, char* str, int key, char* sArr[MAX_DATA_NUM]) {
329     strcpy(str, table_League->data[key - 1]);
330     char* ptr = strtok(str, ",");
331     int i = 0;
332     while (ptr != NULL)
333     {
334         sArr[i] = ptr; //포인터 배열 sArr[i]에 나뉜 ptr의 포인터를 저장하게 된다. table_League의 data형은 앞의 둘과 다르므로 sArr[0]은 key의 값을 저장하고, sArr[1]은 club_key1의 값을 sArr[2]는 club_key2의 값을, sArr[3]에 score의 값이 저장된다
335         ++i;
336         ptr = strtok(NULL, ",");
337     }
338     return;
339 }
340
341

```

특정 문자에 따라 문자열을 나눠주는 함수인 strtok 함수를 사용하여 데이터들이 ‘,’에 따라 나눠도록 함수를 설계하였다. ‘,’에 따라 나뉜 문자들을 ptr에 먼저 저장해놓고, 그 다음 sArr[i]에 저장해주는 방식으로 코드를 설계하였다.

#### 4) 결과화면 분석

set\_table 함수는 txt파일을 쓰기 모드로 여는데 txt파일로부터 데이터를 get\_all\_element를 통해 불러 오려고 할 경우 set\_table 함수와 같이 사용해서는 안 된다. 만약 set\_table 함수를 사용한 뒤, get\_all\_element 함수를 사용하면 set\_table에 의해 변경된 txt 파일로부터 데이터를 얻어오므로 원치 않은 데이터를 불러오게 된다. get\_all\_element 함수를 사용한 뒤 set\_table 함수를 사용해도 같은 문제가 발생하게 된다. 때문에 get\_all\_element 함수와 set\_table 함수는 둘 중 하나만 따로 사용해줘야 한다. 그렇게 실행한 결과는 아래와 같다.

<p>1) set_table</p> <p>2) find_element_by_key와 print_all_table 사용</p>	<pre> 42 int main(void) { 43 44     //초기 설정 단계: Player형 구조체 table_Player의 col 45     Player table_Player = { {0},{0} }; 46     Club table_Club = { {0},{0} }; 47     League table_League = { {0},{0} }; 48 49     Player* Table_p = &amp;table_Player; //table_Player = 50     Club* Table_c = &amp;table_Club; //table_Club 구조체 51     League* Table_l = &amp;table_League; //table_League 52 53     set_table(Table_p, Table_c, Table_l); 54     find_element_by_key(Table_p, Table_c, Table_l); 55     //get_all_element(Table_p, Table_c, Table_l); 56     print_all_table(Table_p, Table_c, Table_l); 57 58     return 0; //main 함수가 정상적으로 종료되었다는 것 59 } 60 </pre>	<pre> 어떤 데이터를 사용하시겠습니까? (table_Player, table_Club, table_League) : table_Player 테이블 구성내용 key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2  어떤 데이터를 찾을 key를 입력하십시오(0=종료): 3 데이터 정보: key,p_name,club_key 선택된 데이터: 3,박주영,2 기입으로 받은 통합정보: 2,명유,1  Player_table key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2  Club_table key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2  League_table key,club_key1,club_key2,score 1,1,2,1:2 2,3,4,2:3 </pre>
		

set\_table는 구조체에 저장된 데이터를 txt파일로 저장해주는 함수이고 위와 같이 main 함수를 적어주었을 때, 프로세스, 메모장에 정상적으로 Player, Club, League 정보가 출력, 저장되는 것을 확인할 수 있다.



	<div data-bbox="311 168 702 593"> <p>Club - Windows 메모장</p> <p>파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)</p> <pre>key,c_name,League_key 1,토트넘,1 2,맨유,1 3,아스날,2 4,맨시티,2 5,레알 마드리드,3 6,바르셀로나,3</pre> </div> <div data-bbox="311 403 702 593"> <p>League - Windows 메모장</p> <p>파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)</p> <pre>key,club_key1,club_key2,score 1,1,2,1:2 2,3,4,2:3 3,5,6,1:1</pre> </div> <div data-bbox="821 168 1460 649"> <p>Player - Windows 메모장</p> <p>파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)</p> <pre>key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2 4,호날두,5 5,메시,6</pre> <pre> 42 int main(void) { 43     //초기 설정 단계: Player형 구조체 table_Player의 column[MAX_COLUMN], data[MAX_DATA_NUM][MAX_DATA_LENGTH]에 모두 0을 채워 44     Player table_Player = { {0}, {0} }; 45     Club table_Club = { {0}, {0} }; 46     League table_League = { {0}, {0} }; 47 48     Player* Table_p = &amp;table_Player; //table_Player 구조체의 시작 주소를 가리키는 포인터 변수 Table_p 선언 49     Club* Table_c = &amp;table_Club; //table_Club 구조체의 시작 주소를 가리키는 포인터 변수 Table_c 선언 50     League* Table_l = &amp;table_League; //table_League 구조체의 시작 주소를 가리키는 포인터 변수 Table_p 선언 51 52     //set_table(Table_p, Table_c, Table_l); 53     get_all_element(Table_p, Table_c, Table_l); 54     print_all_table(Table_p, Table_c, Table_l); 55     find_element_by_key(Table_p, Table_c, Table_l); 56 57     return 0; //main 함수가 정상적으로 종료되었다는 것을 알리기 위해서 사용되었다 58 } 59 60 </pre> </div>
<p>1) get_all_element 함수</p> <p>2) find_element_by_key와 print_all_table 사용</p>	<div data-bbox="311 660 1460 1512"> <p>Microsoft Visual Studio 디버그 콘솔</p> <pre> -----Player_table----- key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2 4,호날두,5 5,메시,6 -----Club_table----- key,c_name,League_key 1,토트넘,1 2,맨유,1 3,아스날,2 4,맨시티,2 5,레알 마드리드,3 6,바르셀로나,3 -----League_table----- key,club_key1,club_key2,score 1,1,2,1:2 2,3,4,2:3 3,5,6,1:1  어떤테이블을 사용하시겠습니까? (table_Player, table_Club, table_League) : table_Player -----테이블 구성내용----- key,p_name,club_key 1,손흥민,1 2,박지성,2 3,박주영,2 4,호날두,5 5,메시,6  연관된 자료를 찾을 key를 입력하십시오(0=종료): 4 테이블 정보: key,p_name,club_key 선택된 데이터: 4,호날두,5 키값으로 찾은 클립정보: 5,레알 마드리드,3  C:\Users\#SDG\source\repos\과제2\Debug\과제2.exe(프로세스 24700개)이(가) 종료되었습니다(코드: 0개). 이 창을 닫으려면 아무 키나 누르세요... </pre> </div>

get\_all\_element 함수는 txt파일에 저장된 정보를 구조체 변수에 삽입해주는 함수이다. 메모장의 내용을 위와 같이 변경해주고 main 함수를 위와 같이 설정해주었을 때, 프로세스에 정상적으로 print\_all\_table 함수가 수행되어 table\_Player엔 4,호날두,5/5,메시,6이 table\_Club엔 5,레알 마드리드,3/6,바르셀로나,3이 table\_League엔 3,5,6,1:1이 구조체 변수에 삽입되어 출력되는 것을 확인해볼 수 있었다.

뿐만 아니라 find\_element\_by\_key 역시 table\_Player에서 key를 4를 입력하였을 때 이에 해당되는 4, 호날두,5가 출력되고 그와 이어지는 club 데이터인 5,레알 마드리드,3가 출력되는 것을 확인해볼 수 있었다. 이를 종합하면 출력 예시의 데이터들이 아닌 다른 값들을 txt 파일에 넣어도 print\_all\_table 함수와 find\_element\_by\_key 함수 둘 다 정상적으로 수행됨을 확인할 수 있었다.