

# Practice in Information Processing - Final Report

Maximilian Fernaldy - C2TB1702

Note: some links and other HTML-related objects may not work in pdf form. Consider reading the webpage format of the report [here](#).

## 3. J\_score2.c

J\_score1.c is fine for a small database like what we have, but it is impractical to copy entire structs three times just to swap their positions. A much more efficient solution is by using an array of pointers. Like the name suggests, an array of pointers is just an array containing the memory addresses of certain variables. In our case, they will correspond to the elements contained in the array storing our data. Since we can access whatever we want inside the data using the pointers, we can simply sort them through their pointers instead, order the pointers accordingly, then after they are sorted, access the data again through the pointer array to display the results, which will now be in order. The original table/array stays the same, but since **the order that they are accessed in** is determined by the position of their respective pointers in the pointer array, we effectively have sorted them without having to move them around.

To make this change, we simply have to create an array of pointers which point to the structs in `table[]` like so:

```
SC *rank_array[number_of_teams]; /* pointer array for sorting */
```

Then, inside `rank_score[]`, before the sorting begins, we initialize the array by filling it with the memory addresses of the corresponding team:

```
void rank_score(SC table[], SC *rank_array[], int number_of_teams)
{
    // Create array of pointers
    for (int i = 0; i < number_of_teams; i++) {
        rank_array[i] = &table[i];
    }
    ...
}
```

Now, the entry `rank_array[i]` is a pointer to the team stored in `table[i]`. We can simply reuse the selection sort code we used in J\_score1.c below the above code block:

```

void rank_score(SC table[], SC *rank_array[], int number_of_teams)
{
    ...
    // Use selection sort
    for (int i = 0; i < number_of_teams-1; i++) {
        int highest_rank_index = i;
        for (int j = i+1; j < number_of_teams; j++) {
            if (rank_array[j]->score > rank_array[highest_rank_index]->score) {
                highest_rank_index = j;
            } else if (rank_array[j]->score == rank_array[highest_rank_index]->score) {
                // Case if a score tie is encountered
                if (rank_array[j]->point_diff > rank_array[highest_rank_index]->point_diff) {
                    // If compared team has larger point difference
                    highest_rank_index = j;
                } else if (rank_array[j]->point_diff == rank_array[highest_rank_index]->point_diff) {
                    // If the point difference is still the same
                    if (rank_array[j]->GF > rank_array[highest_rank_index]->GF) {
                        // If the compared team has more goals scored.
                        highest_rank_index = j;
                    }
                }
            }
        }
    }
    if (highest_rank_index != i) {
        swap_pointers(&rank_array[i], &rank_array[highest_rank_index]);
    }
}
}

```

Of course the `if-else` ladder also stays the same. However, this time, the members are accessed using arrow operators in conjunction with the pointers like so `rank_array[j]->member`. To swap the order of teams, instead of using `swap_SC()`, we create a function `swap_pointers()` to swap their *pointers* around.

## a). Pointers to pointers

Remember that to swap things in an array, we need their memory addresses. That's exactly what we're doing with `swap_pointers()`. We pass the memory addresses of `rank_array[i]` and `rank_array[highest_rank_index]`, which are pointers themselves. This means we're passing *pointers to pointers*, which requires us to use double asterisks when declaring them in the function parameter: `SC **pointerA, SC **pointerB`. Although it may seem confusing, we don't actually have to worry since pointers are also just variables, and exchanging their values are as simple as exchanging the values of any other variable: using the dereference operator (\*).

```

void swap_pointers(SC **pointerA, SC **pointerB) {
    // this function swaps pointers. To do that, we pass in the pointer to the pointer.
    SC *temp = *pointerA;
    *pointerA = *pointerB;
    *pointerB = temp;
}

```

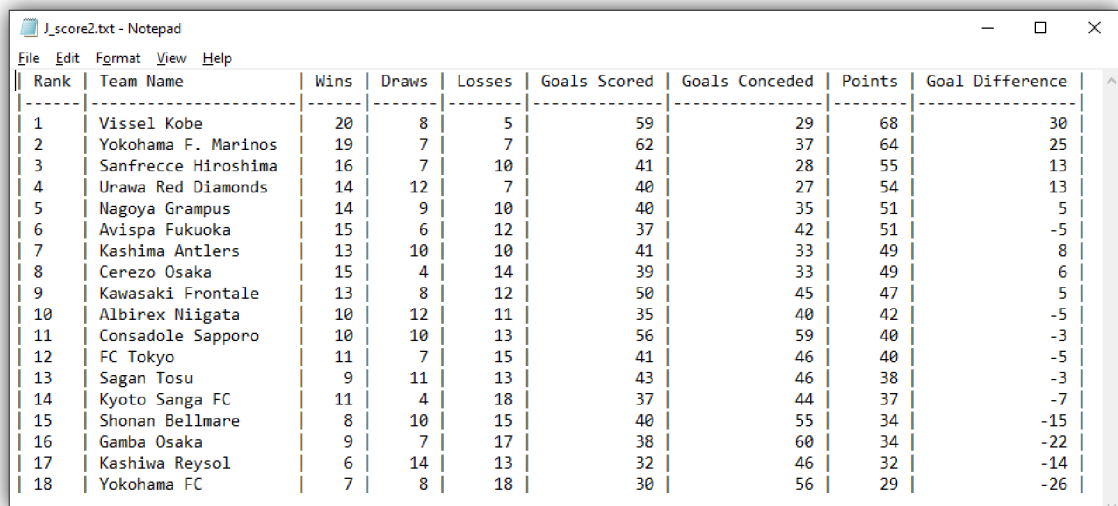
For data types that are not numbers, we have no choice but to use a `temp` variable to store temporary data for swapping. Since a pointer is typically at most only 8 bytes in size, we don't really have to worry about performance here.

## b). Printing the output of J\_score2.c

Printing the output of J\_score2.c is a bit different from J\_score1.c, because what we have sorted is the array of pointers, not the table itself. The table stays as it was, ordered alphabetically. In order to display the ranked data in order, we need to use the array of pointers to access the table. To do this, we use the arrow operator to access the struct members of the variable that is pointed to by `rank_array[i]`. Everything else, including the formatting, stays the same.

```
void write_data(FILE *fout, SC *rank_array[], int number_of_teams)
{
    fprintf(fout, " | Rank | Team Name          | Wins | Draws | Losses | Goals Scored | Goals Conceded | Points | Goal
    fprintf(fout, " |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
    for (int i = 0; i < number_of_teams; i++) {
        fprintf(fout, " | %-4d | %-20s | %4d | %5d | %6d | %12d | %14d | %6d | %15d |\\n",
            i+1,
            rank_array[i]->name,
            rank_array[i]->win,
            rank_array[i]->draw,
            rank_array[i]->loss,
            rank_array[i]->GF,
            rank_array[i]->GA,
            rank_array[i]->score,
            rank_array[i]->point_diff);
    }
}
```

Compiling and running the program, we get an identical result as the one we got from J\_score1.c:



Rank	Team Name	Wins	Draws	Losses	Goals Scored	Goals Conceded	Points	Goal Difference
1	Vissel Kobe	20	8	5	59	29	68	30
2	Yokohama F. Marinos	19	7	7	62	37	64	25
3	Sanfrecce Hiroshima	16	7	10	41	28	55	13
4	Urawa Red Diamonds	14	12	7	40	27	54	13
5	Nagoya Grampus	14	9	10	40	35	51	5
6	Avispa Fukuoka	15	6	12	37	42	51	-5
7	Kashima Antlers	13	10	10	41	33	49	8
8	Cerezo Osaka	15	4	14	39	33	49	6
9	Kawasaki Frontale	13	8	12	50	45	47	5
10	Albirex Niigata	10	12	11	35	40	42	-5
11	Consadole Sapporo	10	10	13	56	59	40	-3
12	FC Tokyo	11	7	15	41	46	40	-5
13	Sagan Tosu	9	11	13	43	46	38	-3
14	Kyoto Sanga FC	11	4	18	37	44	37	-7
15	Shonan Bellmare	8	10	15	40	55	34	-15
16	Gamba Osaka	9	7	17	38	60	34	-22
17	Kashiwa Reysol	6	14	13	32	46	32	-14
18	Yokohama FC	7	8	18	30	56	29	-26