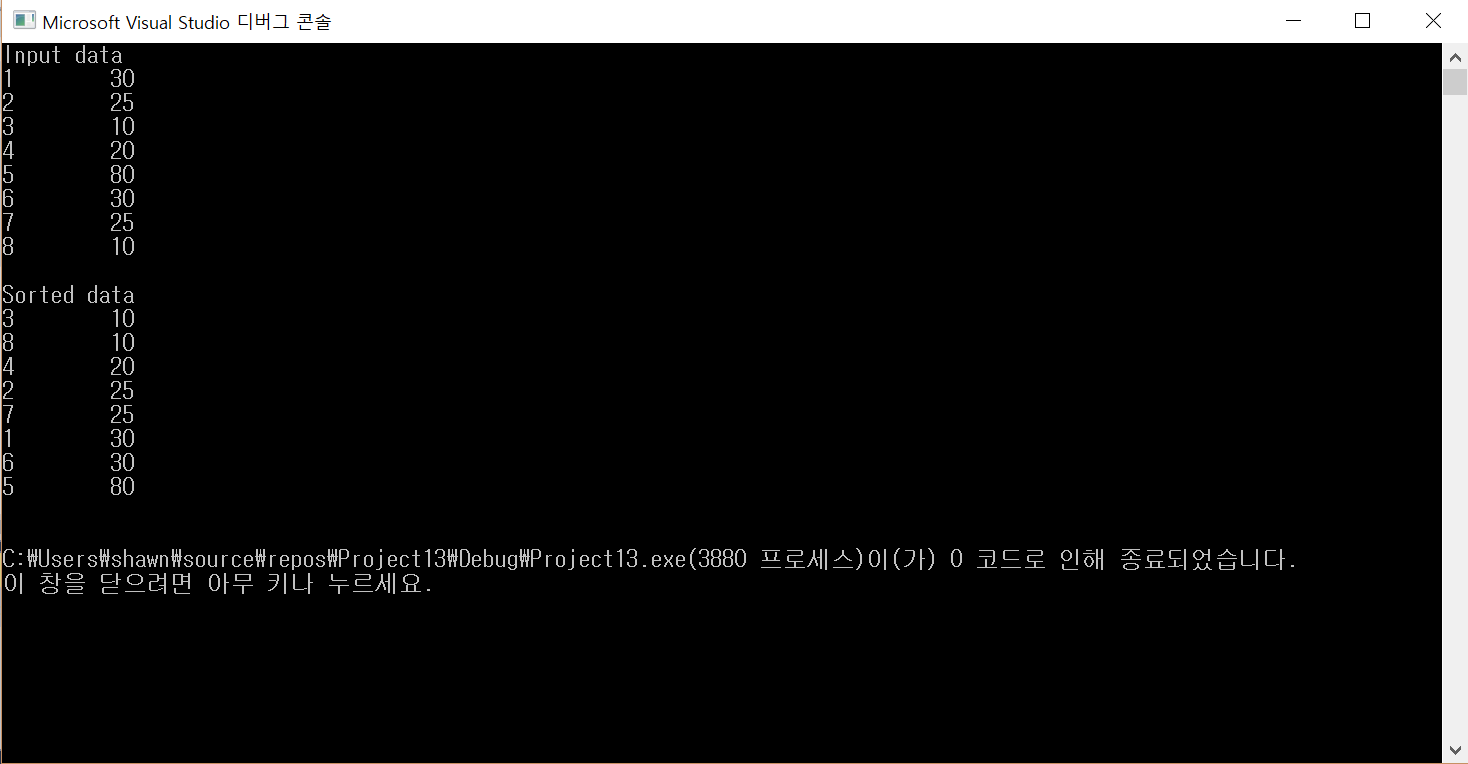
1. 코드 분석 및 주석
2. #include "stdlib.h"
3. #include "stdio.h"
4. #include "string.h"
5. typedef struct data {
6. int \*id;
7. int \*score;
8. } data;
9. #define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )
10. void selection\_sort(data \*list, int n)
11. {
12. int i, j, least, temp;
13. for (i = 0; i < n - 1; i++) {
14. least = i;
15. for (j = i + 1; j < n; j++)
16. if (list->score[j] < list->score[least]) least = j;
17. SWAP(list->score[i], list->score[least], temp);
18. SWAP(list->id[i], list->id[least], temp);
19. }
20. }
21. void selection\_sort\_stable(data \*list, int n)
22. {
23. int i, j, least, temp;
24. int add\_mem;//addtional memory를 저장할 공간
25. for (i = 0; i < n - 1; i++) {
26. least = i;
27. for (j = i + 1; j < n; j++)
28. if (list->score[j] < list->score[least]) least = j;
29. //현재 list->score[least]에 최솟값이 저장. 이걸 addtional memory에 넣음.
30. add\_mem = list->score[least];
31. //unstable 이 생기는 이유는 score[i-1]에서 같은 value가 있는데 id가 새로 추가될 value보다 더 클 경우이다.
32. if (i > 0 && list->score[i - 1] == add\_mem) {
33. if (list->id[i - 1] > list->id[least]) {
34. SWAP(list->score[i-1], list->score[least], temp);
35. SWAP(list->id[i-1], list->id[least], temp);
36. }
37. }
38. SWAP(list->score[i], list->score[least], temp);
39. SWAP(list->id[i], list->id[least], temp);
40. }
41. }
42. int main()
43. {
44. int data\_id[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
45. int data\_score[] = { 30, 25, 10, 20, 80, 30, 25, 10 };
46. int in\_size = 8;
47. data \*input = (data \*)malloc(sizeof(data));
48. input->id = data\_id;
49. input->score = data\_score;
50. //Print out the input data
51. printf("Input data\n");
52. for (int i = 0; i < in\_size; i++)
53. printf("%d\t %d\n", input->id[i], input->score[i]);
54. printf("\n");
55. // Produce the stable sorting results by replacing 'selection\_sort' with 'selection\_sort\_stable'.
56. selection\_sort\_stable(input, in\_size);
57. //Print out the sorted data
58. printf("Sorted data\n");
59. for (int i = 0; i < in\_size; i++)
60. printf("%d\t %d\n", input->id[i], input->score[i]);
61. printf("\n");
62. return 0;
63. }

2. 결과 및 콘솔 창



3. 코드 분석

Selection sorting이 unstable 한 이유는 swap이 바로 인접한 공간에서 이루어지는 것이 아니라 간격을 두고도 이루어 질 수 있기 때문이다. 예를 들어서 이 예제의 list에 25는 총 2번째 index, 7번째 index에 두 번 있다. 하지만 2번째 index의 25는 8번째 index의 10과 swap을 하기 때문에 8번째 index에 자리잡게 된다. 따라서 7번째 index 보다 더 뒷 순위로 밀리게 된다. 이걸 해결하기 위해서는 우선 additional memory를 선언하여 swap을 두 번 진행해줘야 한다.

순서대로 정렬을 하다 보면 10,10,20,25(7번째 인덱스),30 …여기까지 정렬이 진행되었을 것이다. 이 때 30과 바꿀 25는 2번째의 index 이다. 하지만 30과 바로 바꾸기 전에 30의 전 25와 같은지 비교를 해야 한다. 만약 새로 정렬될 데이터와 같고, 새로 정렬된 index가 이미 정렬된 index보다 적다면 이 둘을 먼저 swap 해줘야 한다. 그러고 다면 index 2 25가 있었던 자리에 index 7 25가 오게 되고 이 때 30과 바꿔주면 가능하다.

if (i > 0 && list->score[i - 1] == add\_mem) {

if (list->id[i - 1] > list->id[least]) {

SWAP(list->score[i-1], list->score[least], temp);

SWAP(list->id[i-1], list->id[least], temp);

}

}