REPORT

Data Structures

Student ID : 1771008
Name : Minjeong Kim

Homework7_1

변수(variable) 분석

data	
Type	**name**
int*	id
int*	score

data_id 배열에 대한 주소를 data타입의 id 요소에 저장한다.
data_score 배열에 대한 주소를 data타입의 score 요소에 저장한다.

함수(function) 분석

void selection_sort_stable(data *inList, data* outList, int n)
stable = 같은 숫자가 나왔을 때 그 숫자의 index 순서가 바뀌면 안된다.
따라서 swap을 이용하지 않고, sorting이 되지 않은 숫자들 중에서 가장 작은 값을 다른 list에 저장하는 방법을 사용한다. 이때, sorting이 완료된 숫자 같은 경우 NULL을 설정해서 구분한다.
따라서 이를 그림으로 만들어보면 아래와 같다.

윗 줄은 inList / 아랫 줄은 outList 이다.

1 : 30	2 : 25	3 : 10	4 : 20	5 : 80	6 : 30	7 : 25	8 : 10

1 : 30	2 : 25	3 : null	4 : 20	5 : 80	6 : 30	7 : 25	8 : 10
3 : 10							

1 : 30	2 : 25	3 : null	4 : 20	5 : 80	6 : 30	7 : 25	8 : null
3 : 10	8 : 10						

1 : 30	2 : 25	3 : null	4 : null	5 : 80	6 : 30	7 : 25	8 : null
3 : 10	8 : 10	4 : 20					

1 : 30	2 : null	3 : null	4 : null	5 : 80	6 : 30	7 : 25	8 : null
3 : 10	8 : 10	4 : 20	2 : 25				

1 : 30	2 : null	3 : null	4 : null	5 : 80	6 : 30	7 : null	8 : null
3 : 10	8 : 10	4 : 20	2 : 25	7 : 25			

1 : null	2 : null	3 : null	4 : null	5 : 80	6 : 30	7 : null	8 : null
3 : 10	8 : 10	4 : 20	2 : 25	7 : 25	1 : 30		

1 : null	2 : null	3 : null	4 : null	5 : 80	6 : null	7 : null	8 : null
3 : 10	8 : 10	4 : 20	2 : 25	7 : 25	1 : 30	6 : 30	

1 : null	2 : null	3 : null	4 : null	5 : null	6 : null	7 : null	8 : null
3 : 10	8 : 10	4 : 20	2 : 25	7 : 25	1 : 30	6 : 30	5 : 80

매 반복마다. score가 가장 작은 index를 min으로 설정한다. 이때, index 순서가 그대로 갈 수 있는 이유는 첫번째 for문에서의 반복이 1로 시작해서 n으로 끝나기 때문이다.

전체 코드(full code)

```c
// selection_sort.cpp : Defines the entry point for the console application.
//

#include "stdlib.h"
#include "stdio.h"
#include "string.h"

typedef struct data {
    int *id;
    int *score;
} data;

 #define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )

void selection_sort(data *list, int n)
{
    int i, j, least, temp;
    for (i = 0; i < n - 1; i++) {
        least = i;
        for (j = i + 1; j<n; j++)
            if (list->score[j]<list->score[least]) least = j;
        SWAP(list->score[i], list->score[least], temp);
        SWAP(list->id[i], list->id[least], temp);
    }
}

void selection_sort_stable(data *inList, data* outList, int n)
{
    int min;
    for (int i = 0; i < n; i++) {
        min = 0;
        while (inList->score[min] == NULL) {
            min += 1;
        }
        for (int j = 0; j < n; j++) { //min 찾기
            if (inList->score[j]!=NULL && (inList->score[min] > inList->score[j]))
                min = j;
        }
        outList->id[i] = inList->id[min];
        outList->score[i] = inList->score[min];
        inList->score[min] = NULL;
    }
}

int main()
{
    int data_id[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
    int data_score[] = { 30, 25, 10, 20, 80, 30, 25, 10 };

    int in_size = 8;
    data *input = (data *)malloc(sizeof(data));
    input->id = data_id;
    input->score = data_score;

    data *output = (data *)malloc(sizeof(data));
    int out_id[8], out_score[8];
    output->id = out_id;
    output->score = out_score;

    //Print out the input data
    printf("Input data\n");
    for (int i = 0; i < in_size; i++)
        printf("%d\t %d\n", input->id[i], input->score[i]);
    printf("\n");

    // Produce the stable sorting results by replacing 'selection_sort' with 'selection_sort_stable'.
    //selection_sort(input, in_size);

    selection_sort_stable(input, output, in_size);

    //Print out the sorted data
    printf("Sorted data\n");
    for (int i = 0; i < in_size; i++)
        printf("%d\t %d\n", output->id[i], output->score[i]);
    printf("\n");

    return 0;
}
```

Homework7_2

변수(variable) 분석

element	
Type	**name**
int	num
int [ ]	d

BUCKETS 10 : 한자리당 가능한 숫자의 개수 (0-9)
DIGITS 4 : 자리수의 개수 (0~9999)
RAND_MAX 9999 : 최대 숫자 (0~9999)
SIZE 1000` : 정렬을 할 숫자의 개수

변수 저장 예시
숫자 : 1234
num = 1234
d[3] = 1 / d[2] = 2 / d[1] = 3 / d[0] = 4

이 하나의 element에 대한 배열을 만든다. 이때 배열사이즈는 정렬을 할 숫자의 개수인 SIZE이다.

함수(function) 분석

void print (element* arr)
element 배열에 올바른 값이 들어갔는지를 확인하기 위해서, print 함수를 만들었다.

void make_rand_number(element input[])
0~9999까지의 랜덤한 숫자 1000개를 생성해낸다. 이때 숫자 생성과 더불어 element구조체에 각 값들을 설정하고, 그 객체의 값을 element 배열에 넣는 작업도 같이 하기위해, input[] array를 인자로 받아 이 input[]에 random값을 저장한다. 이때 element 구조체의 요소인 d[] 배열에도 자리수마다 각각 저장하기 위해 for문을 한 번 더 사용해서, 각 자리수마다 저장하는 연산을 한번 더 했다.

void counting_sort(element input[], cons tint numdigit, element result[])
배열 input과 result가 주어져있는데, input을 counting sort한 결과를 result에 저장한다. counting_sort의 과정은 총 4가지 단계로 이루어져 있다.

1) 자리수 (0~9)의 배열 생성과 초기화 (for문을 사용해서 각 자리를 0으로 초기화한다.)
2) 자리수의 빈도수를 digit[] 배열에 저장한다. : input의 numdigit(1234의 0번째 index 등의 의미)에 해당하는 숫자를 보고, 해당하는 숫자의 값을 digit[] 배열의 index의 관점에서 생각한다. 그 후 해당 index의 값을 1 씩 증가시킨다면, 자리수의 빈도수가 digit[] 배열에 저장되게 된다.
3) 빈도수를 누적방식으로 나타내기위해 현재 digit = 현재 digit + 이전 digit 의 연산을 한다.
4) input의 가장 마지막 index(9999)에서 0까지 접근한다. 해당 input의 numdigit자리의 숫자가 가진 누적 빈도수를 digit에서 확인하고, 해당 digit[] 배열에 들어있는 값을 result[] 배열의 index로 설정하고, 누적 digit의 값을 1 감소한다.

void radix_sort(element input[])

radix_sort의 핵심은 numdigit즉, counting sort가 이러날 digit의 자리수가 0에서부터 3까지 LSD에서 MSD로 증가한다는 점에 있다. 따라서 counting_sort의 numdigit 값을 0에서부터 3까지 차례대로 설정해준후, 최종 결과를 print 함수를 이용해서 정렬된 결과를 출력한다.

전체 코드(full code)

```c

#include "stdlib.h"
#include "stdio.h"
#include "math.h"

 #define BUCKETS 10
 #define DIGITS 4
 #define RAND_MAX 9999
 #define SIZE 1000

typedef struct {
    int num;
    int d[DIGITS]; //d3 d2 d1 d0 순서이다.
}element;



void print(element* arr) {
    for (int i = 0; i < SIZE; i++) {
        printf("%d :", arr[i].num);
        printf("%d ", arr[i].d[3]);
        printf("%d ", arr[i].d[2]);
        printf("%d ", arr[i].d[1]);
        printf("%d \n", arr[i].d[0]);
    }
}

void make_rand_number(element input[]) {
    for (int i = 0; i < SIZE; i++) {
        int num = rand() % (RAND_MAX + 1) + 1;
        input[i].num = num;
        for (int j = 0; j < DIGITS; j++) {
            input[i].d[j] = num % 10;
            num /= 10;
        }
    }
}

void counting_sort(element input[], const int numdigit, element result[]) {

    int digit[BUCKETS];

    for (int i = 0; i < BUCKETS; i++) {
        digit[i] = 0;
    }

    for (int i = 0; i < SIZE; i++) {
        int index = input[i].d[numdigit];
        digit[index] += 1;
    }


    for (int i = 1; i < BUCKETS; i++) {
        digit[i] = digit[i] + digit[i - 1];
    }

    for (int i = (SIZE - 1); i >= 0; i--) {
        int index = input[i].d[numdigit];
```

```
59              int c_index = digit[index];
60              result[c_index - 1] = input[i];
61              digit[index] -= 1;
62          }
63
64      }
65
66      void radix_sort(element input[]) {
67          element result_a[SIZE];
68          element result_b[SIZE];
69
70          counting_sort(input, 0, result_a);
71          counting_sort(result_a, 1, result_b);
72          counting_sort(result_b, 2, result_a);
73          counting_sort(result_a, 3, result_b);
74
75          printf("===\n");
76          print(result_b);
77      }
78
79      int main() {
80          element input[SIZE];
81          make_rand_number(input);
82          print(input);
83          printf("====\n");
84
85          radix_sort(input);
86
87          return 0;
88      }
```