# Assignment4 Report

20181016 KwonYongmin

## Assignment4.c

To make heap memory allocator, I manage two variables.

One is “heap_area” which is char array with 64 bytes, allocated in heap memory. And I make the integer variable “filled” represent the index of filled “heap_area” continuously.

The other one is “mem_table” which is an array consisted with 64 “mem_table_entry”s. “mem_table_entry” is a structure defined in “assignment4.h”. It contains order of data, each data’s name, and size. Similar with “heap_area”, I also manage the integer variable “current_table_index” which means the number of data in “mem_table_entry”.

```c
#include "assignment4.h"

int main (){

    char* heap_area = (char *)malloc(sizeof(char) * 64);
    int filled = 0; // index of filled heap area
    mem_table_entry mem_table[64];
    int current_table_index = 0; // number of variables

    manage_heap_area(heap_area, filled, mem_table, current_table_index);

    return 0;
}
```

**Figure 1 main function**

## Assignment4.h

There are two structures and one function.

**mem_table_entry**

As I mentioned above, it represents the name and size of the data which is allocated to “heap_area”.

```c
typedef struct {
    char name[256];
    int size;
}mem_table_entry;
```

**Figure 2 structure of “mem_table_entry”**

```
if(strncmp(type, "short", 5) == 0){
    size = 2;
    if(filled + size <= 64){
        mem_table_entry new_entry;
        strcpy(new_entry.name, name);
        new_entry.size = size;
        mem_table[current_table_index] = new_entry;
        current_table_index++;
        printf("Please input a value for the data type\n");
        short data;
        scanf("%hi", &data);
        void* ptr = &data;
        char* ptr2 = ptr;
        for(int i = 0; i < sizeof(short); i++){
            heap_area[filled + i] = ptr2[i];
        }
        filled += size;
```

**Figure 3 usage of "mem_table_entry"**

**struct_table_entry**

This structure is for supporting structure to our heap allocator. When allocating structure in "heap_area", I make an array of "struct_table_entry". And using data in this array, it allocates the data in the structure to "heap area".

```
typedef struct {
    char type[8];
    char data[1024];
}struct_table_entry;
```

**Figure 4 structure of "struct_table_entry"**

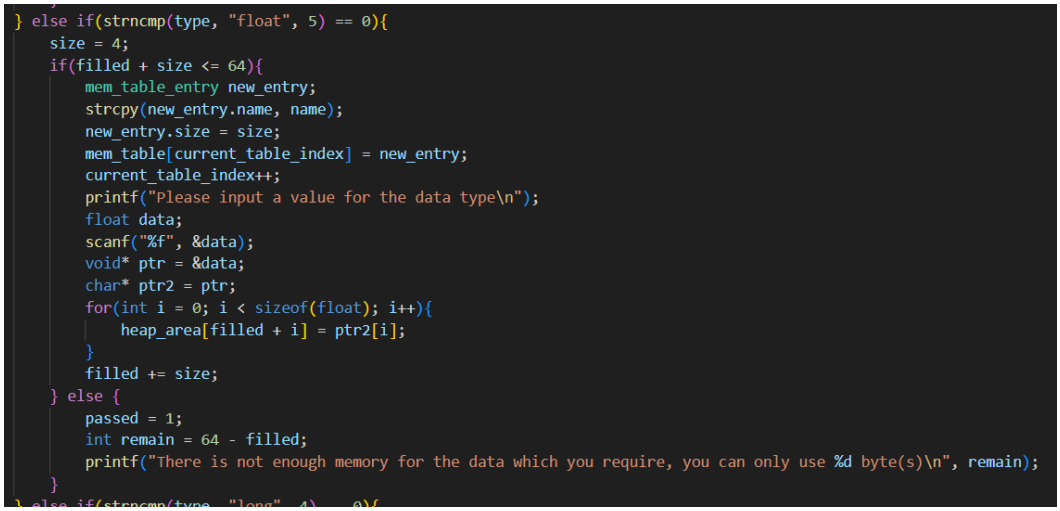
```
else if(strncmp(type, "struct", 6) == 0){
  unsigned int number_of_data;
  printf("How many data should be in the struct\n");
  scanf("%u", &number_of_data);
  struct_table_entry struct_table[number_of_data];
  printf("Please input each type and its value\n");
  int type_error = 0;
  for(unsigned int i = 0; i < number_of_data; i++){
      scanf("%s %s", struct_table[i].type, struct_table[i].data);
      if(strncmp(struct_table[i].type, "short", 5) == 0){
          size += 2;
      }else if(strncmp(struct_table[i].type, "char", 4) == 0){
          size += 1;
      }else if(strncmp(struct_table[i].type, "float", 5) == 0){
          size += 4;
      }else if(strncmp(struct_table[i].type, "long", 4) == 0){
          size += 8;
      } else {
          type_error++;
      }
  }
```

**Figure 5 usage of "struct_table_entry"**

### manage_heap_area

This function is quite long, but it works easy. It conducts two processes, allocation and deallocation to “heap_area”.

In data allocation process, it just takes the information of data from stdin, and if the information is correct, it allocates the data to “heap_area” and saves the information of the data to “mem_table”.



```
} else if(strncmp(type, "float", 5) == 0){
    size = 4;
    if(filled + size <= 64){
        mem_table_entry new_entry;
        strcpy(new_entry.name, name);
        new_entry.size = size;
        mem_table[current_table_index] = new_entry;
        current_table_index++;
        printf("Please input a value for the data type\n");
        float data;
        scanf("%f", &data);
        void* ptr = &data;
        char* ptr2 = ptr;
        for(int i = 0; i < sizeof(float); i++){
            heap_area[filled + i] = ptr2[i];
        }
        filled += size;
    } else {
        passed = 1;
        int remain = 64 - filled;
        printf("There is not enough memory for the data which you require, you can only use %d byte(s)\n", remain);
    }
```

**Figure 6 allocation process**

Deallocation process is simple. Using “mem_table” it finds the offset and index of target variable in “heap_area”. And then it clears target data from both of “heap_area” and “mem_table”.

```
} else if(task_type == 2){
    printf("Input the name of data you want to deallocate\n");
    char target_variable[256];
    scanf("%s", target_variable);
    int target_variable_size = 0;
    int target_variable_offset = 0;
    int target_variable_index = 0;
    int founded = 0;
    for(int i = 0; i < current_table_index; i++){
        if(founded == 0){
            if(strncmp(mem_table[i].name, target_variable, 256) == 0){
                target_variable_size = mem_table[i].size;
                target_variable_index = i;
                founded = 1;
            } else {
                target_variable_offset += mem_table[i].size;
            }
        }
    }

    if(founded == 1){
        for(int j = 0; j < filled - (target_variable_offset + target_variable_size); j++){
            heap_area[target_variable_offset + j] = heap_area[target_variable_offset + target_variable_size + j];
        }
        for(int k = 1; k <= target_variable_size; k++){
            heap_area[filled - k] = 0;
        }
        filled -= target_variable_size;

        for(int l = 0; l < current_table_index - (target_variable_index + 1); l++){
            mem_table[target_variable_index + l] = mem_table[target_variable_index + 1 + l];
        }
        current_table_index--;
    }
```

**Figure 7 deallocation process**