First, I defined opt to be 1 or 2 (allocate and deallocate)

Case allocate:

```
    type, name os type ← stdin
    switch type
        case invalid:
            error message, no operation
        case struct:
if (meta.type == struct_t) {
    int n;
    printf("How many data should be in the struct\n");
    assert(scanf("%d", &n) == 1);
    getchar(); //read new line
    if (n>8 || n<1){ //check condition
        printf("Struct should have at most 8 types and at least 1 type\n");
        continue;
    }
    printf("Please input type each type and its value \n");
    meta.start = data_ptr; //set data pointer to be the start position of the type
    size_t original_ptr = data_ptr; //ser original data pointer = data_pointer to return to for
avoiding dump memory (invalid struct case)
    int valid = 1; //boolean to check if struct data is valid
    for (int i=0; i<n; i++) {
        char line[1000]; //store input lines
        assert(fgets(line, sizeof(line), stdin));
        char* t = strtok(line, " "); //struct data type
        char* val = strtok(NULL, " "); //struct data value
        val[strcspn(val, "\n")] = 0; //exclude the new line
        enum type sub_type = define_type(t);
        if (sub_type== invalid) { //check for invalid type
            printf("Invalid type\n");
            valid = 0;
            break;
        }
        if(!check_range(sub_type, val)) { //check for invalid value
            printf("There is invalid input\n");
            valid = 0;
            break;
        }
//handle each type and value
        if (sub_type == char_t) { //case char
            if((data_ptr+1) > DATA_SIZE) { check if out of memory
                printf("There is not enough memory for the data you require, you can
only use %zu byte(s)\n", DATA_SIZE - data_ptr);
                valid = 0;
                break;
        }
            unsigned char v = (unsigned char)val[0];
            data[data_ptr++] = v; //update data pointer in memory dump
        } else if (sub_type == short_t) {
            if((data_ptr+ sizeof(unsigned short)) > DATA_SIZE) {
```

```
                        printf("There is not enough memory for the data you require, you can
only                use %zu byte(s)\n", DATA_SIZE - data_ptr);
                        valid = 0;
                        break;
                }
                unsigned short v = (unsigned short)strtoul(val, NULL, 10);
                memcpy(&data[data_ptr], &v, sizeof(v));
                data_ptr += sizeof(v);
        } else if (sub_type == int_t) { //case int
                if((data_ptr+ sizeof(unsigned int)) > DATA_SIZE) {
                        printf("There is not enough memory for the data you require, you can
only                use %zu byte(s)\n", DATA_SIZE - data_ptr);
                        valid = 0;
                        break;
                }
                unsigned int v = (unsigned int)strtoul(val, NULL, 10);
                memcpy(&data[data_ptr], &v, sizeof(v));copy memory of value to data[]
                data_ptr += sizeof(v); //data pointer increment
        } else if (sub_type == long_t) { //case long
                if((data_ptr+ sizeof(unsigned long long) > DATA_SIZE)) {
                        printf("There is not enough memory for the data you require, you can
only use %zu byte(s)\n", DATA_SIZE - data_ptr);
                        valid = 0;
                        break;
                }
                unsigned long long v = strtoull(val, NULL, 10);
                memcpy(&data[data_ptr], &v, sizeof(v));
                data_ptr += sizeof(v);
        } else if (sub_type == float_t) { //case float
                if((data_ptr+ sizeof(float)) > DATA_SIZE) {
                        printf("There is not enough memory for the data you require, you can
only use %zu byte(s)\n", DATA_SIZE - data_ptr);
                        valid = 0;
                        break;
        }
                float v = strtof(val, NULL);
                memcpy(&data[data_ptr], &v, sizeof(v));
                data_ptr += sizeof(v);
        }
    }
    if (!valid) { //if invalid
    memset(&data[original_ptr], 0, data_ptr – original_ptr); //set memory dump of struct to be 0
    data_ptr = original_ptr;  //set data pointer to original_ptr( befrore the invalid struct is written
to memory)
    continue;
}

case else (not struct) //the logic is as same as for struct type
```

Case Deallocate

```
else if (opt == 2) { //deallocate
    char remove[100];
}
    printf("Input the name of data you want to deallocate \n");
    assert(scanf("%s", remove) == 1);
    getchar();
    int found = 0;
    for (int i=0; i<count; i++) {
              if (strcmp(metadata[i].name, remove)==0) {
              found = 1;
              int start = metadata[i].start;  //initialize start as the pointer to start of the remove
data
              int stop = metadata[i].stop; //initialize stop as the pointer to stop of remove data
              int size = stop-start; //size of the removed data

    if(stop ==data_ptr) { //last in heap
              memset(&data[start], 0 , size); //if removed data is the last in heap then set the data
to be 0 in heap
              data_ptr = start;
    }
else {
    int n_shift = data_ptr – stop; //number of shifts need to make
    memmove(&data[start], &data[stop], n_shift); move n bytes (number of shifts) to the start
point of removed data
    memset(&data[data_ptr-size], 0 , size); //set the remaining part after moving the
remaining data to be 0 in heap
    for (int j=i+1; j<count; j++) {
              metadata[j].start -=size; //set the start and stop of data again after deallocating by
size (size of removed data)
              metadata[j].stop -= size;
    }
    data_ptr -= size; //set the pointer by size (size of removed data)
}
    for (int j=i; j < count - 1; j++)
              metadata[j] = metadata[j + 1]; //move the index of other data after the removed
data
              count--;
              printf("%s has been deallocated\n", remove);
              break;
}
}
    if (!found) {
              printf("Deallocating wrong data\n");
}
    printf("There is memory dump!\n");
    dump_mem(data, DATA_SIZE);

    printf("\n-----Data you have now-----\n");
    for (int i = 0; i < count; i++) {
              printf("%s \n", metadata[i].name);
}
```