```c
typedef struct queue_node {
    int id;
    int location;
    int mem_size;
    struct queue_node *next;
    struct queue_node *prev;
}tQueueNode;

typedef struct {
    tQueueNode *front;
    tQueueNode *rear;
    int count;
    int total_size;
}tQueue;
```

tQueue struct 裡面多設一個參數 total_size，用來存目前 memory（mask）的使用量，至於 tQueueNode 裡面參數不變。

從 main function 開始介紹

```c
if (operation == 1){
    printf("  enter id:");
    scanf("%d", &id);
    printf("  specify data type (units) you want:");
    scanf("%d", &mem_size);
    ret = enqueue_node(queue, id, mem_size);

    if (ret == 0){
        printf("    Cannot enter to the queue  \n");
    }
    print_buffer_status();
}
```

在 operation 於 1 時，與先前的作業相同做 enqueue 的部分，只是這次多傳入一個 mem_size 的參數，此參數是使用者輸入這次需要使用多少記憶體空間，
那傳入 enqueue 之後也跟之前的作業相同，進入到 our_malloc 去要記憶體位置，這裡也跟先前的差不多，只是多一個使用者輸入的記憶體使用量而已，
而這次的插入方式我選擇使用 first-fit memory allocation 的方式，所以在尋找記憶體位置的時候就比較簡單去想，直接尋找連續的記憶體空間大小剛好符合輸入的記憶體空間即可。

```
bool buf_mask[NUM_BYTE_BUF] = {0};
```

這此為了要擁有無限大的 mask_buf，我使用了 bool array 來解決此問題，會使用 bool 是因為 mask 只需存 0、1 兩個值，所以用 bool 即可。

```
else if (operation == 2){
    printf ("  enter an ID to remove ");
    scanf("%d", &id);
    node = NULL;
    node = find_target_node(queue, id);
    if (node == NULL){
        printf ("   cannot find the target node\n");
    }
    else{
        dequeue_node(queue, node);
    }
    print_buffer_status();

}
```

再來是 operation 於 2 時，就去做 dequeue 的動作，刪除節點，釋放記憶體位置（mask_buf），刪除 link-list 的 node 節點並釋放其空間，這也與先前的大同小異，也只是差在多一個記憶體使用空間的參數而已。