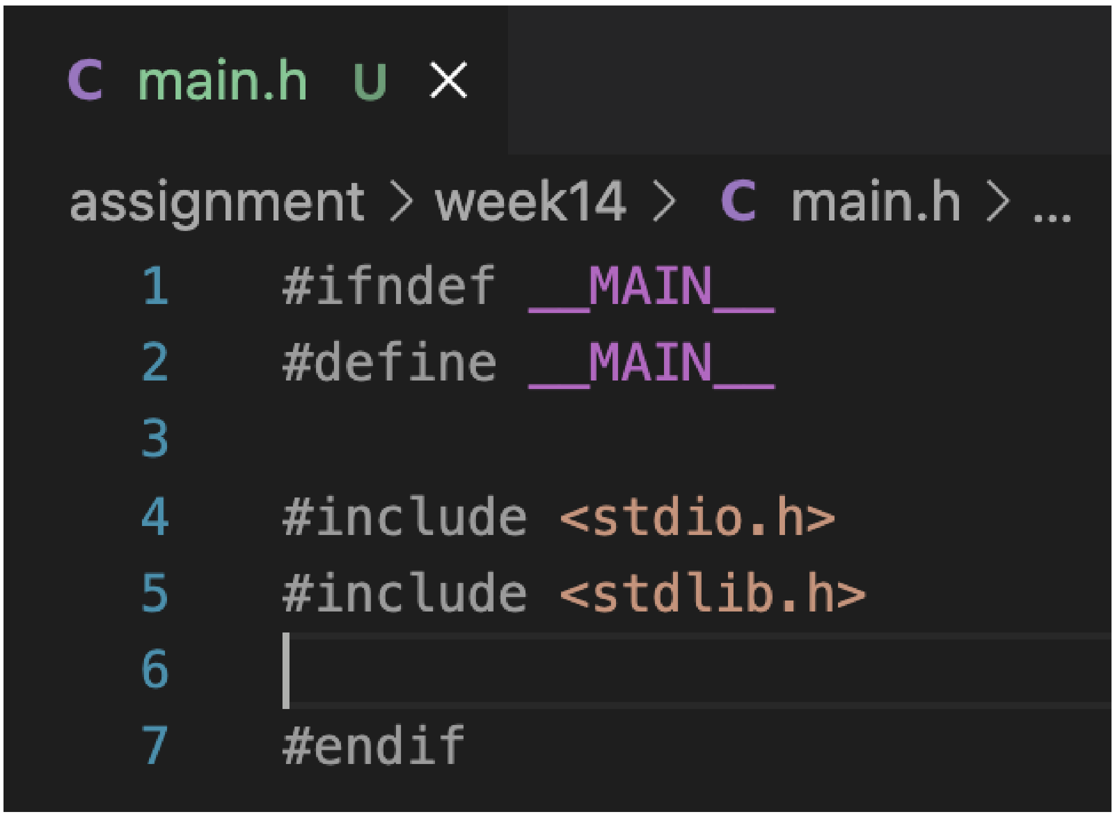
電子碩一 113368080 張富順

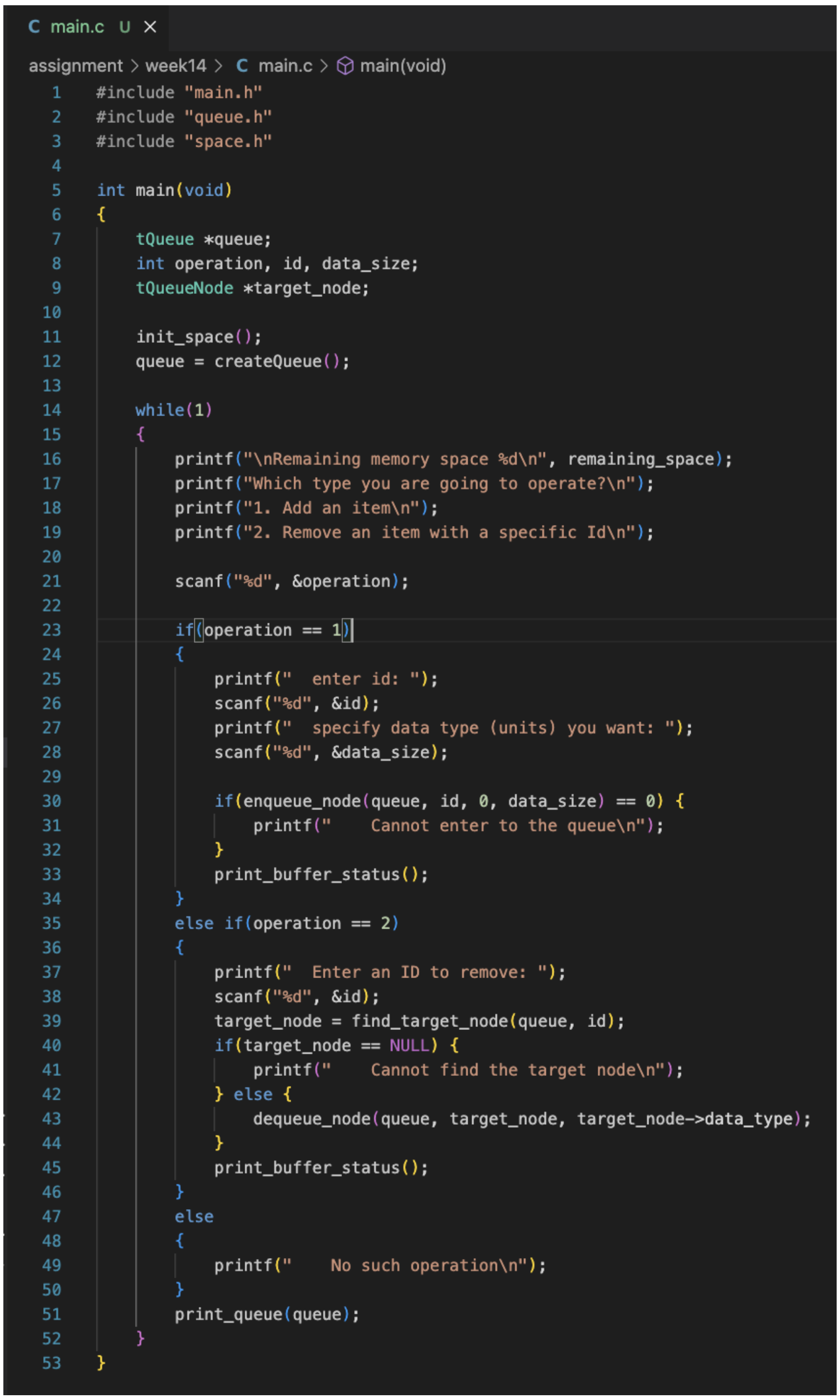
mian.h:



```c
#ifndef __MAIN__
#define __MAIN__

#include <stdio.h>
#include <stdlib.h>

#endif
```

mian.c:

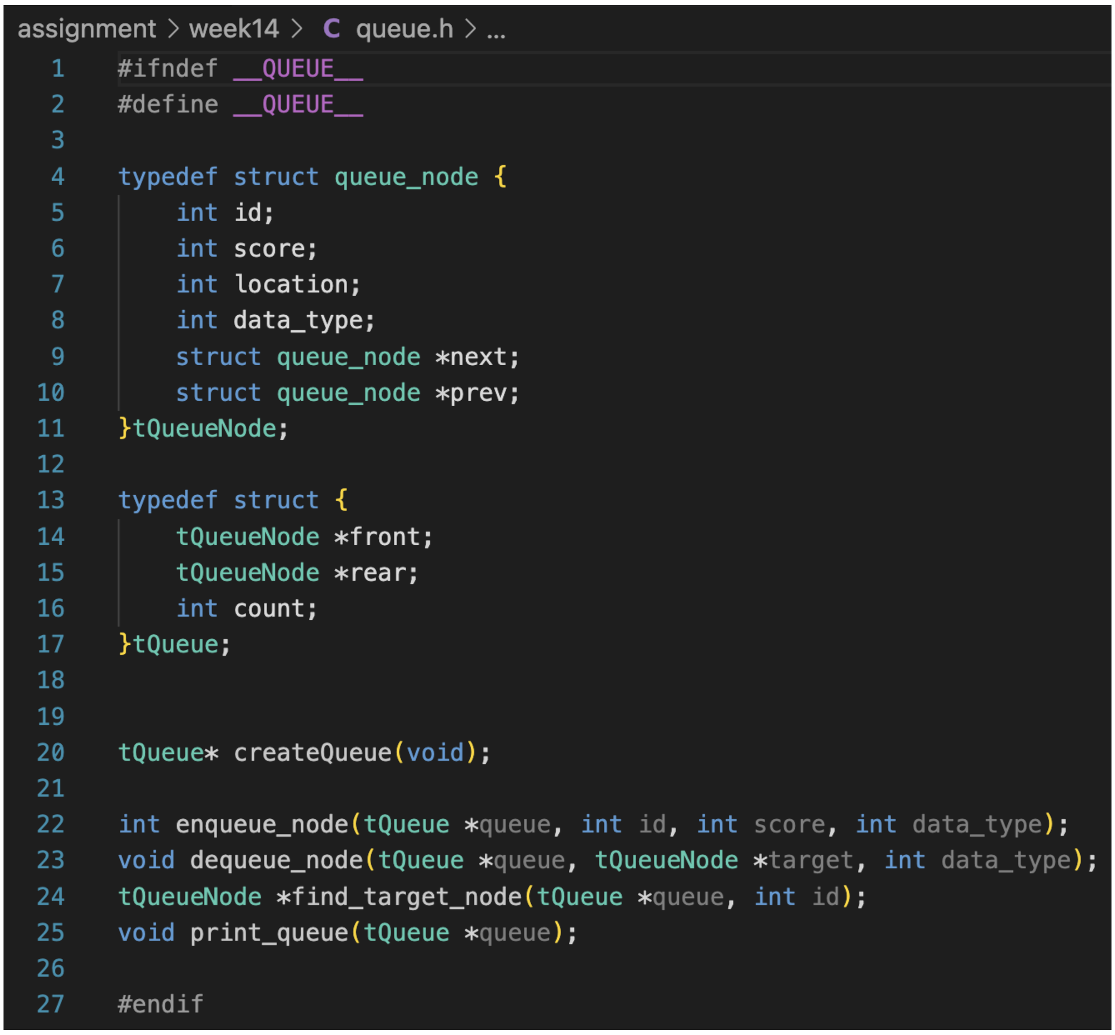
共有兩種操作模式，新增與移出特定元素。當 operation 等於 1 時，判斷是否可以插入元素。而當 operation 等於 2 時，則判斷是否有符合條件的元素可以移除。



```c
#include "main.h"
#include "queue.h"
#include "space.h"

int main(void)
{
    tQueue *queue;
    int operation, id, data_size;
    tQueueNode *target_node;

    init_space();
    queue = createQueue();

    while(1)
    {
        printf("\nRemaining memory space %d\n", remaining_space);
        printf("Which type you are going to operate?\n");
        printf("1. Add an item\n");
        printf("2. Remove an item with a specific Id\n");

        scanf("%d", &operation);

        if(operation == 1)
        {
            printf("  enter id: ");
            scanf("%d", &id);
            printf("  specify data type (units) you want: ");
            scanf("%d", &data_size);

            if(enqueue_node(queue, id, 0, data_size) == 0) {
                printf("    Cannot enter to the queue\n");
            }
            print_buffer_status();
        }
        else if(operation == 2)
        {
            printf("  Enter an ID to remove: ");
            scanf("%d", &id);
            target_node = find_target_node(queue, id);
            if(target_node == NULL) {
                printf("    Cannot find the target node\n");
            } else {
                dequeue_node(queue, target_node, target_node->data_type);
            }
            print_buffer_status();
        }
        else
        {
            printf("    No such operation\n");
        }
        print_queue(queue);
    }
}
```

queue.h:

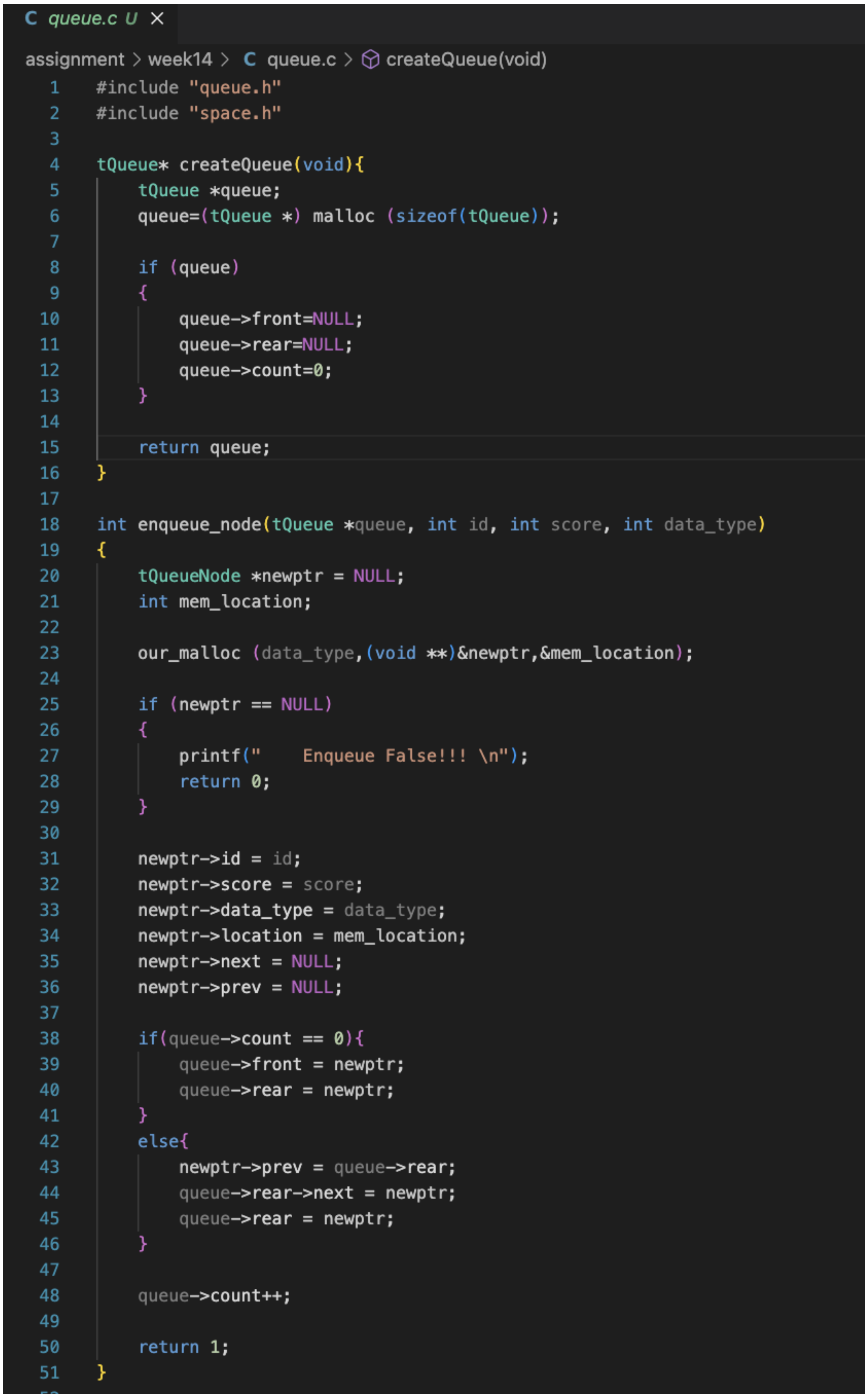
定義資料結構以及 Function 有哪些，其中 queue_node->data_type 為輸入的元素大小


assignment > week14 > C queue.h > ...

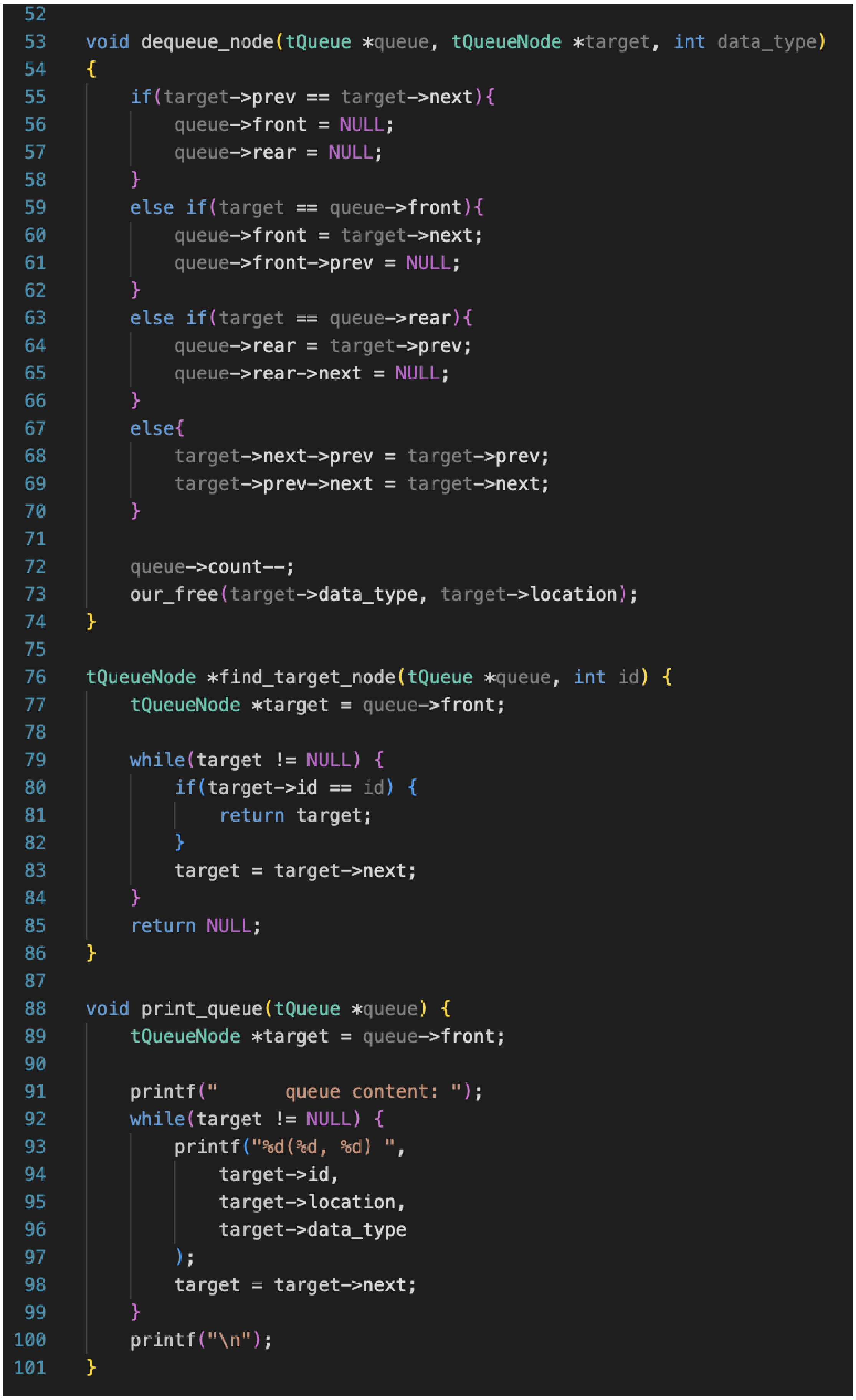
```c
#ifndef __QUEUE__
#define __QUEUE__

typedef struct queue_node {
    int id;
    int score;
    int location;
    int data_type;
    struct queue_node *next;
    struct queue_node *prev;
}tQueueNode;

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


tQueue* createQueue(void);

int enqueue_node(tQueue *queue, int id, int score, int data_type);
void dequeue_node(tQueue *queue, tQueueNode *target, int data_type);
tQueueNode *find_target_node(tQueue *queue, int id);
void print_queue(tQueue *queue);

#endif
```

queue.c:

- createQueue:建立空的 queue_node
- enqueue_node:插入 queue_node，並透過 our_malloc 去分配 queue_node 的記憶體位址
- dequeue_node:移除特定位置的 queue_node，並透過 our_free 釋放 queue_node 的記憶體位址
- *find_target_node:去 queue 中找尋有沒有符合條件的 id 元素
- print_queue:列印所有在 queue_node 中的元素



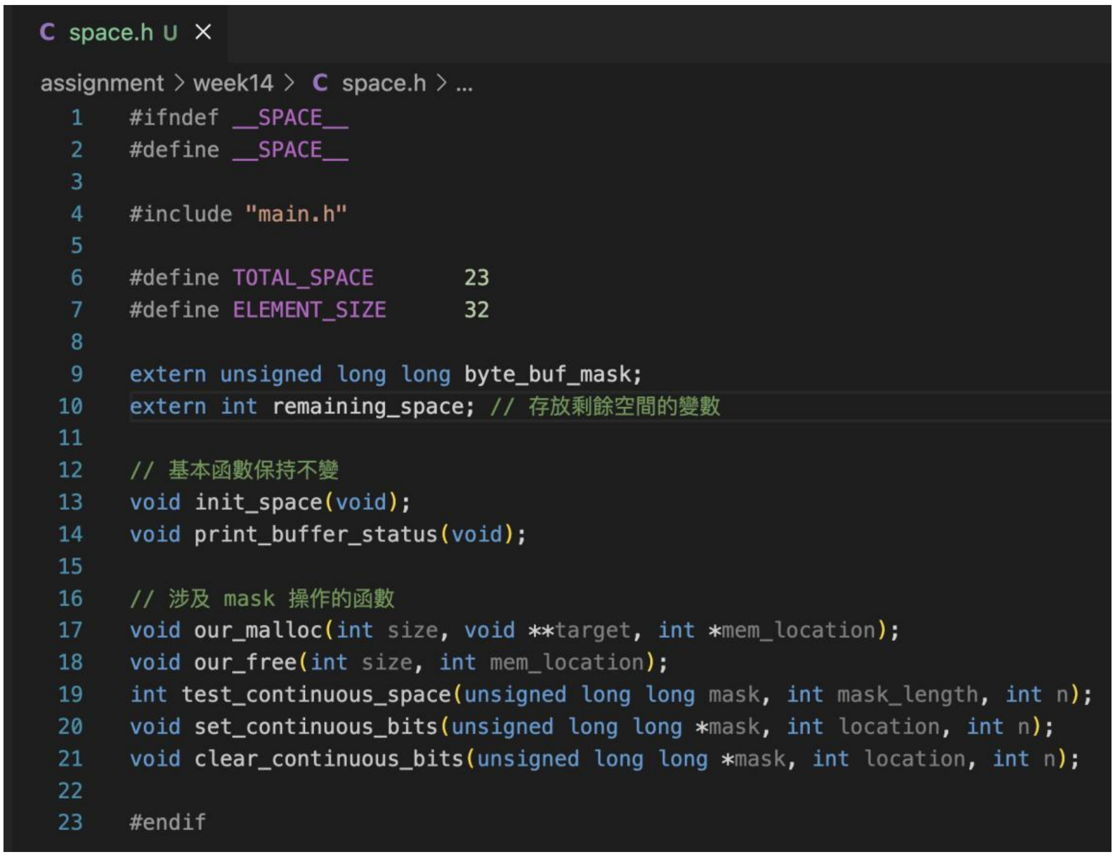
```c
#include "queue.h"
#include "space.h"

tQueue* createQueue(void){
    tQueue *queue;
    queue=(tQueue *) malloc (sizeof(tQueue));

    if (queue)
    {
        queue->front=NULL;
        queue->rear=NULL;
        queue->count=0;
    }

    return queue;
}

int enqueue_node(tQueue *queue, int id, int score, int data_type)
{
    tQueueNode *newptr = NULL;
    int mem_location;

    our_malloc (data_type,(void **)&newptr,&mem_location);

    if (newptr == NULL)
    {
        printf("    Enqueue False!!! \n");
        return 0;
    }

    newptr->id = id;
    newptr->score = score;
    newptr->data_type = data_type;
    newptr->location = mem_location;
    newptr->next = NULL;
    newptr->prev = NULL;

    if(queue->count == 0){
        queue->front = newptr;
        queue->rear = newptr;
    }
    else{
        newptr->prev = queue->rear;
        queue->rear->next = newptr;
        queue->rear = newptr;
    }

    queue->count++;

    return 1;
}
```



```c

void dequeue_node(tQueue *queue, tQueueNode *target, int data_type)
{
    if(target->prev == target->next){
        queue->front = NULL;
        queue->rear = NULL;
    }
    else if(target == queue->front){
        queue->front = target->next;
        queue->front->prev = NULL;
    }
    else if(target == queue->rear){
        queue->rear = target->prev;
        queue->rear->next = NULL;
    }
    else{
        target->next->prev = target->prev;
        target->prev->next = target->next;
    }

    queue->count--;
    our_free(target->data_type, target->location);
}

tQueueNode *find_target_node(tQueue *queue, int id) {
    tQueueNode *target = queue->front;

    while(target != NULL) {
        if(target->id == id) {
            return target;
        }
        target = target->next;
    }
    return NULL;
}

void print_queue(tQueue *queue) {
    tQueueNode *target = queue->front;

    printf("      queue content: ");
    while(target != NULL) {
        printf("%d(%d, %d) ",
            target->id,
            target->location,
            target->data_type
        );
        target = target->next;
    }
    printf("\n");
}
```

space.h:

remaining_space 負責記錄在 buffer 中可用的記憶體空間



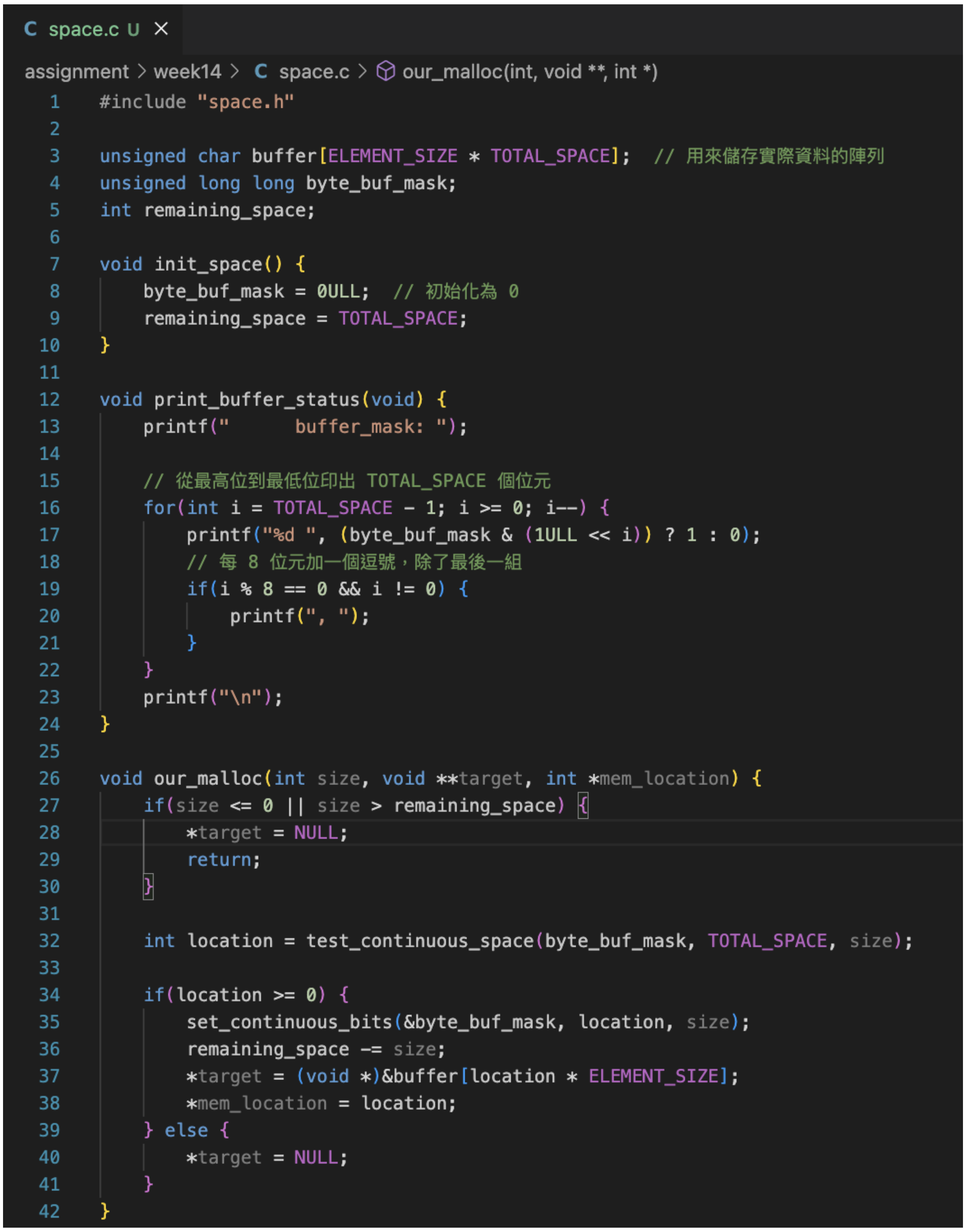
C space.h U ×

assignment > week14 > C space.h > ...

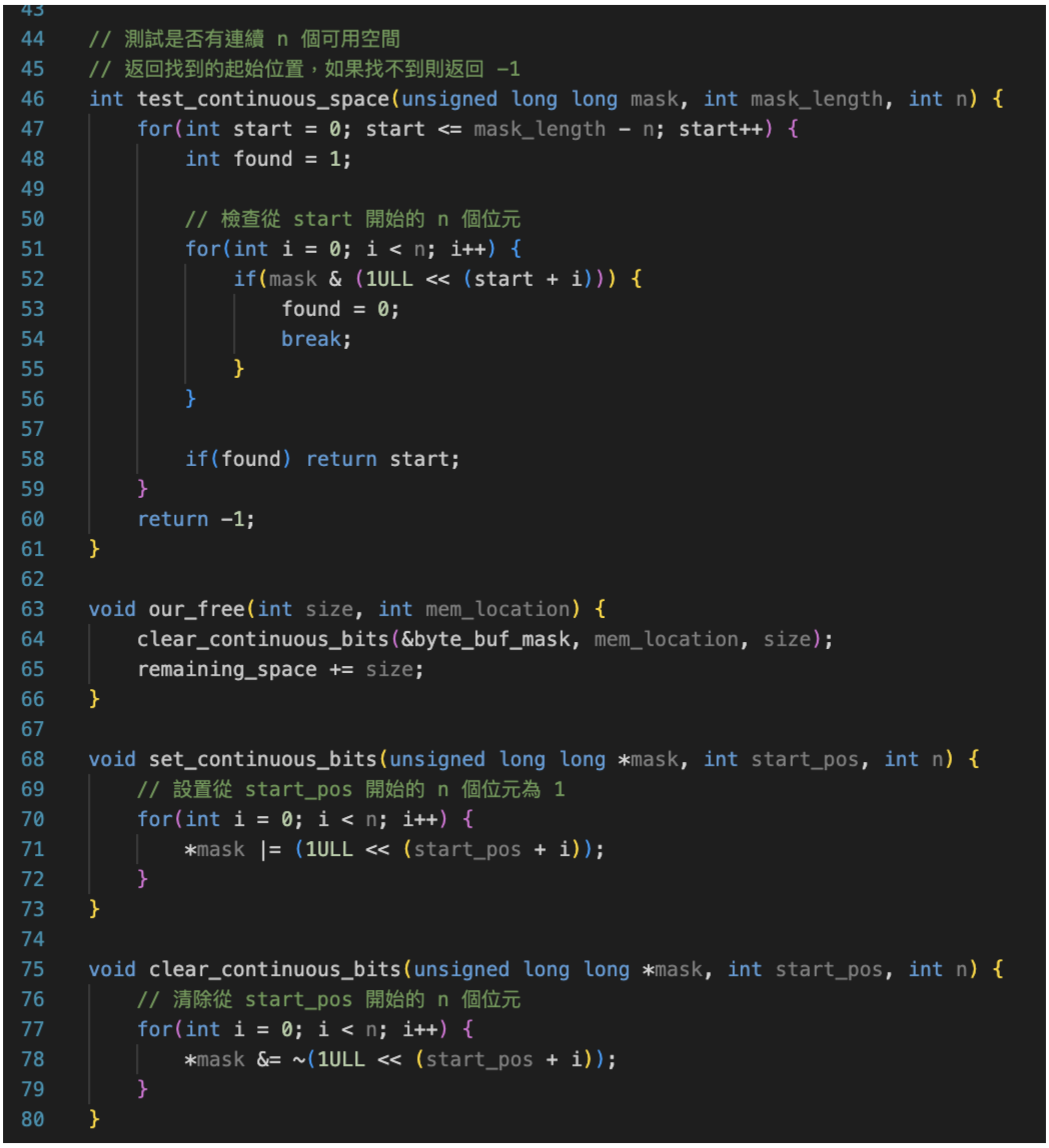
```c
#ifndef __SPACE__
#define __SPACE__

#include "main.h"

#define TOTAL_SPACE       23
#define ELEMENT_SIZE      32

extern unsigned long long byte_buf_mask;
extern int remaining_space; // 存放剩餘空間的變數

// 基本函數保持不變
void init_space(void);
void print_buffer_status(void);

// 涉及 mask 操作的函數
void our_malloc(int size, void **target, int *mem_location);
void our_free(int size, int mem_location);
int test_continuous_space(unsigned long long mask, int mask_length, int n);
void set_continuous_bits(unsigned long long *mask, int location, int n);
void clear_continuous_bits(unsigned long long *mask, int location, int n);

#endif
```

space.c:

- init_space:初始化 mask 和 remaining_space
- print_buffer_status:印出當前 buffer 內記憶體佔用的情況
- our_malloc:實際分配 buffer 記憶體給 queue_node，location 若大於等於 0 則配置記憶體空間
- test_continuous_space:測試是否有連續 n 個可用空間進行 buffer 的記憶體分配
- our_free:釋放在 buffer 中分配的空間
- set_continuous_bits:設置 mask 的連續的 bits 為 1
- clear_continuous_bits: 設置 queue 中符合移除條件的 mask 設為 0


C space.c U ×
assignment > week14 > C space.c > our_malloc(int, void **, int *)

```c
#include "space.h"

unsigned char buffer[ELEMENT_SIZE * TOTAL_SPACE];  // 用來儲存實際資料的陣列
unsigned long long byte_buf_mask;
int remaining_space;

void init_space() {
    byte_buf_mask = 0ULL;  // 初始化為 0
    remaining_space = TOTAL_SPACE;
}

void print_buffer_status(void) {
    printf("      buffer_mask: ");

    // 從最高位到最低位印出 TOTAL_SPACE 個位元
    for(int i = TOTAL_SPACE - 1; i >= 0; i--) {
        printf("%d ", (byte_buf_mask & (1ULL << i)) ? 1 : 0);
        // 每 8 位元加一個逗號，除了最後一組
        if(i % 8 == 0 && i != 0) {
            printf(", ");
        }
    }
    printf("\n");
}

void our_malloc(int size, void **target, int *mem_location) {
    if(size <= 0 || size > remaining_space) {
        *target = NULL;
        return;
    }

    int location = test_continuous_space(byte_buf_mask, TOTAL_SPACE, size);

    if(location >= 0) {
        set_continuous_bits(&byte_buf_mask, location, size);
        remaining_space -= size;
        *target = (void *)&buffer[location * ELEMENT_SIZE];
        *mem_location = location;
    } else {
        *target = NULL;
    }
}
```



```c
// 測試是否有連續 n 個可用空間
// 返回找到的起始位置，如果找不到則返回 -1
int test_continuous_space(unsigned long long mask, int mask_length, int n) {
    for(int start = 0; start <= mask_length - n; start++) {
        int found = 1;

        // 檢查從 start 開始的 n 個位元
        for(int i = 0; i < n; i++) {
            if(mask & (1ULL << (start + i))) {
                found = 0;
                break;
            }
        }

        if(found) return start;
    }
    return -1;
}

void our_free(int size, int mem_location) {
    clear_continuous_bits(&byte_buf_mask, mem_location, size);
    remaining_space += size;
}

void set_continuous_bits(unsigned long long *mask, int start_pos, int n) {
    // 設置從 start_pos 開始的 n 個位元為 1
    for(int i = 0; i < n; i++) {
        *mask |= (1ULL << (start_pos + i));
    }
}

void clear_continuous_bits(unsigned long long *mask, int start_pos, int n) {
    // 清除從 start_pos 開始的 n 個位元
    for(int i = 0; i < n; i++) {
        *mask &= ~(1ULL << (start_pos + i));
    }
}
```