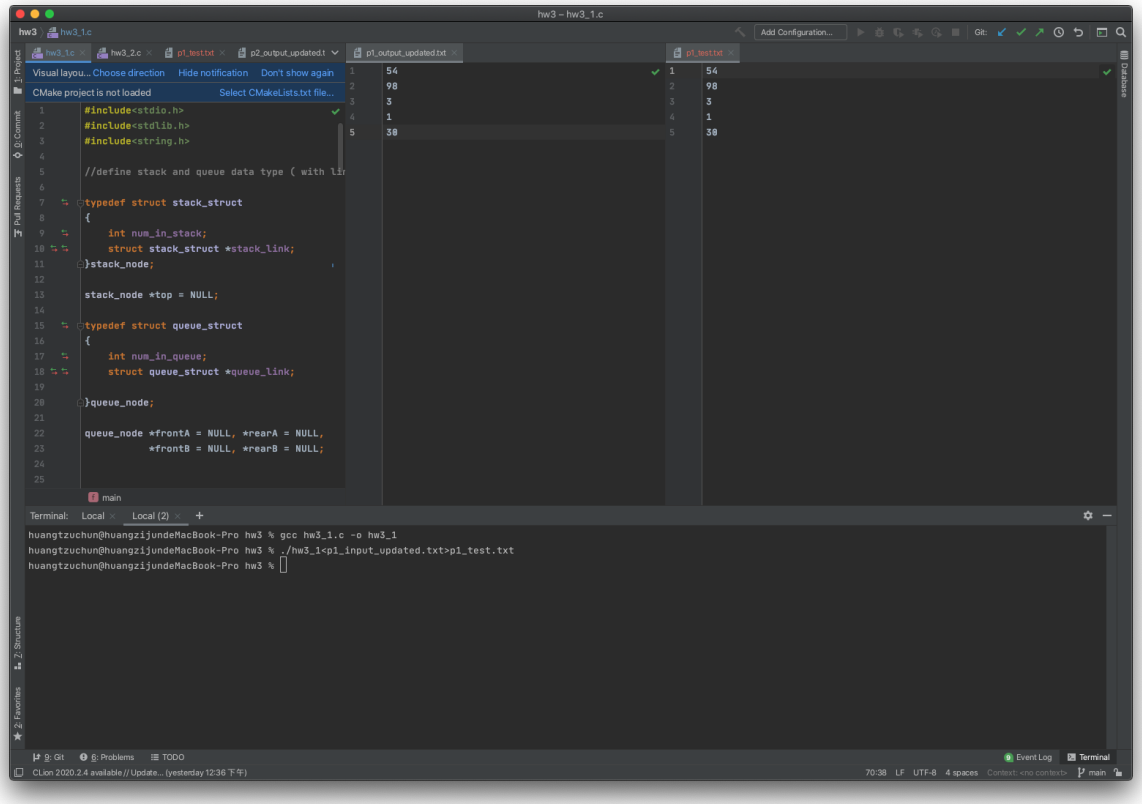
HW3 README

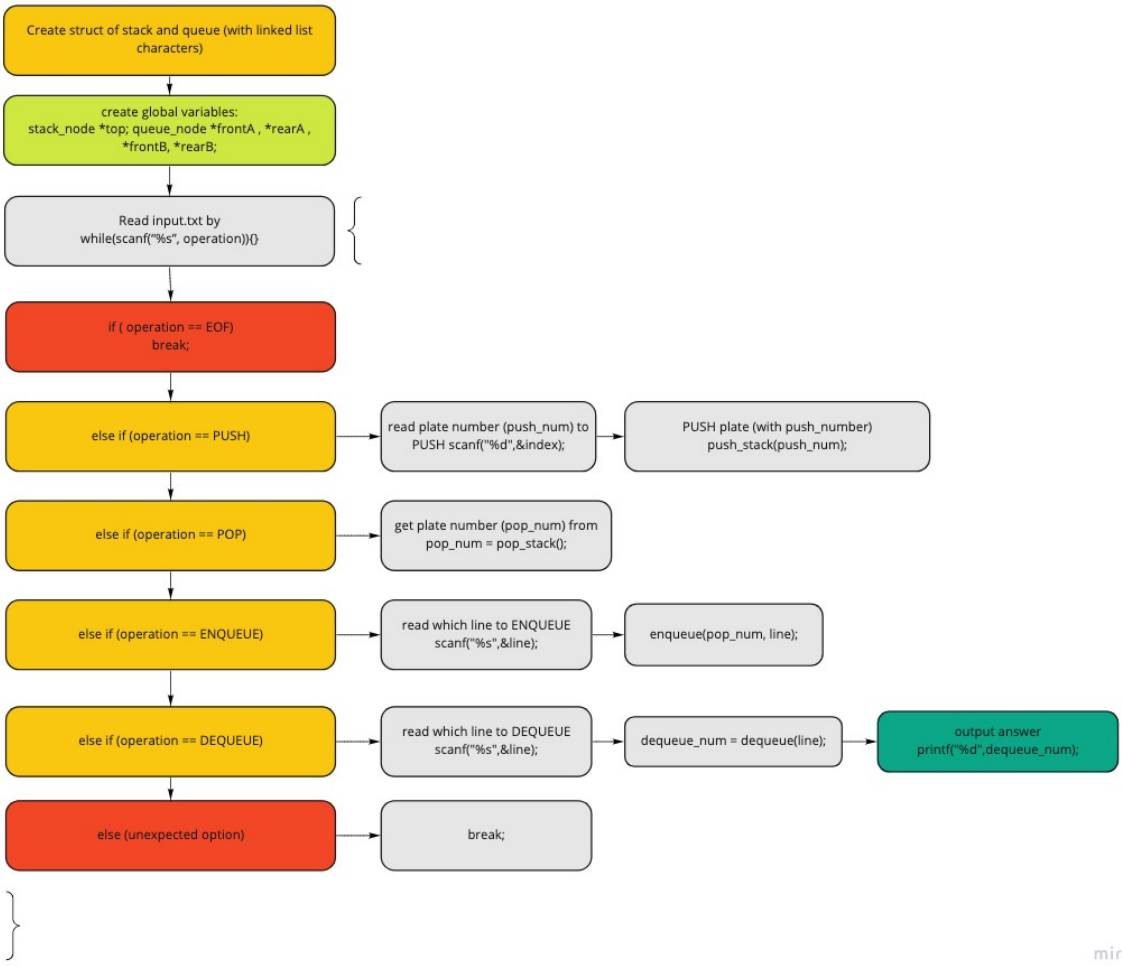
E14084117, CSIE112, 資訊二乙, 黃子峻

Part 1 Queueing in campus cafeteria

(1)Result screenshot:



(2) Program architecture:



(3)Program structs:

Struct for stack (with linked list characters)

```
typedef struct stack_struct
{
    int num_in_stack;
    struct stack_struct *stack_link;
}stack_node;

stack_node *top = NULL;
```

- `num_in_stack` stores the random number, which is assigned to the plate.
- Set `*top` as global variable.

Struct for Queue (with linked list characters)

```
typedef struct queue_struct
{
    int num_in_queue;
    struct queue_struct *queue_link;

}queue_node;

queue_node *frontA = NULL, *rearA = NULL,
           *frontB = NULL, *rearB = NULL;
```

- `num_in queue` is the same idea as `num_in_stack` mention above.
- Set `*frontA`, `*frontB`, `*rearA`, `*rearB` as global parameters.

---

(4) Program functions:

Stack functions

`void push_stack(int push_num)`

> Push the a plate (with push_num N) which refills by the staff into the plate stack.

Usage

- `push_stack(push_num);`

Parameters

- `int push_num` : the plate number on the plate refilled by the staff into the plate stack.

Return values

- no return values (void)

Code

```
void push_stack(int push_num)
{
    stack_node *new_stack_node = (stack_node*)malloc(sizeof(stack_node));
```

```
    new_stack_node -> num_in_stack = push_num;

    if (top != NULL)
        new_stack_node->stack_link = top;

    top = new_stack_node;
    return;
}
```

`int pop_stack(void)`

> Indicates the customer takes a plate from the top of the plate stake.

Usage

- `pop_num = pop_stack();`

Return values

- `int pop_num` : `int` type plate number which is removed from the stack.

Code

```
int pop_stack(void)
{
    if (top == NULL){
        //printf("The stack is empty\n");
        return -1;
    }

    stack_node *pop_stack_node = top;

    int pop_num = pop_stack_node->num_in_stack;
    top = pop_stack_node ->stack_link;

    free(pop_stack_node);

    return pop_num;
}
```

Queue functions

`void enqueue(int enqueue_num, char line)`

> indicates a customer joins the end of the line A or B.

Usage

- `enqueue(enqueue_num, line);`

Parameters

- `char line` : `'A'` or `'B'` to indicate which line to join.
- `int enqueue_num` : the plate number we want to enqueue to the `line` (A or B) queue, which gets from `pop_stack()`.

Return values

- no return values (void)

Code

```
void enqueue(int enqueue_num, char line)
{
    queue_node *new_queue_node = (queue_node *) malloc(sizeof(queue_node));
    new_queue_node->num_in_queue = enqueue_num;
    new_queue_node->queue_link= NULL;

    switch (line){
        case 'A':
            if(frontA)
                rearA->queue_link = new_queue_node;
            else
                frontA = new_queue_node;
            rearA = new_queue_node;
            break;

        case 'B':
            if(frontB)
                rearB->queue_link = new_queue_node;
            else
                frontB = new_queue_node;
            rearB = new_queue_node;
            break;
    }
    return;
}
```

`int dequeue(char line)`

> indicates a customer at the front of the line X leaves the line to checkout.

Usage

- `dequeue_num = dequeue(line);`

Parameters

- `char line` : `'A'` or `'B'` to indicate which line to leave.

Return values

- `int dequeue_num` : plate number of the plate which leaves from the line (queue).

Code

```
int dequeue(char line){
    queue_node *out_queue_node;
    int out_value;
    switch (line) {
        case 'A':
            out_queue_node = frontA;
            if(!out_queue_node)
            {
                printf("queue is empty\n");
                return -1;
            }
            out_value = out_queue_node->num_in_queue;
            frontA = out_queue_node ->queue_link;
            break;
        case 'B':
            out_queue_node = frontB;
            if(!out_queue_node)
            {
                printf("queue is empty\n");
                return -1;
            }
            out_value = out_queue_node->num_in_queue;
            frontB = out_queue_node ->queue_link;
            break;
    }
    free(out_queue_node);
    return out_value;
}
```

---

(5) How I design my program :

How I read the input data and turn it to different operations?

- For the `./a.out<input.txt>output.txt` command, we can use `scanf("%s", operation)` to get input letters in to `char *operation`.
- Using `while(scanf("%s", operation)!=EOF){}` loop to read input letter by letter, it `break` when read to `EOF` (end of file).
- Using `if(strcmp(operation, "OPERATION"))` to decide which `OPERATION` to do .(`OPERATION` can be `PUSH`, `POP`, `ENQUEUE` or `DEQUEUE`.

The advantages of using linked list to built the stack and queue.

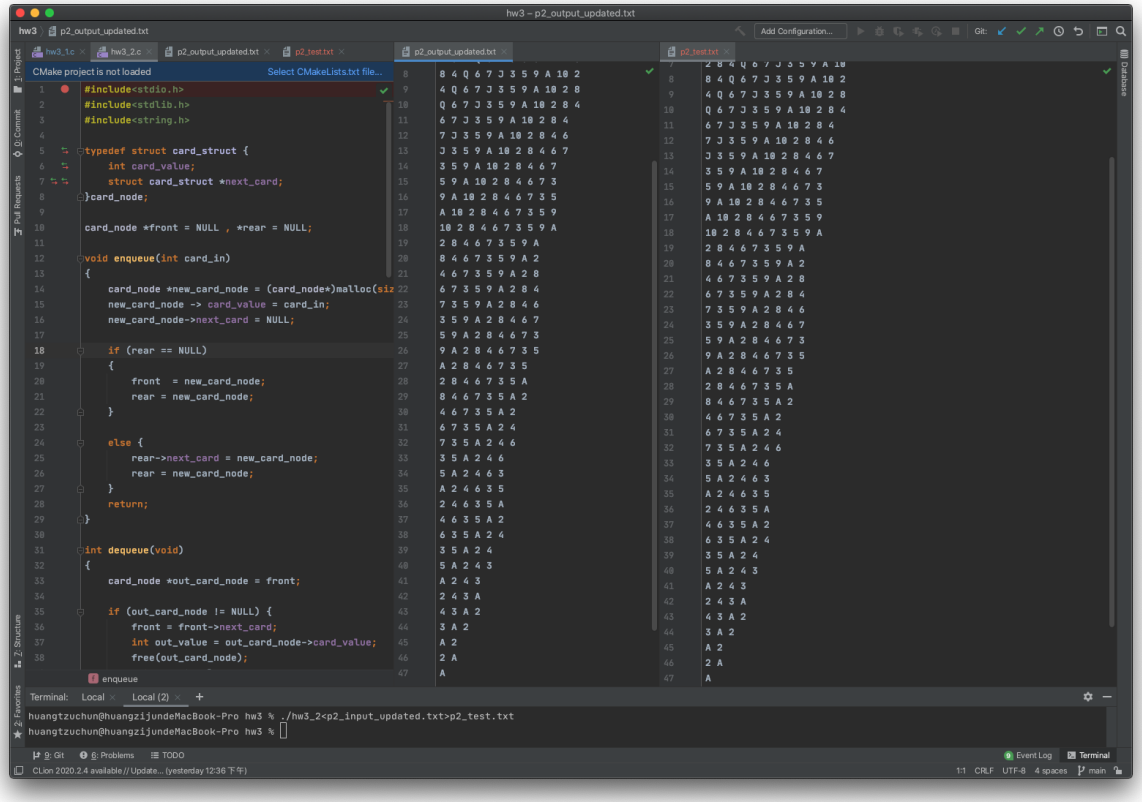
- Using linked list to build the stack and queue , it's not need to considering whether the stack and queue is full or not any times when you adding data.

How I simplified this program compare to last time?

- By using global variables : `*top`, `*frontA`, `*frontB`, `*rearA`, `*rearB`, I don't need to think about how to passing those pointer to the functions.

Part 2 Solitaire

(1)Result screenshot:



(2) Program architecture:

(3)Program structs:

```
typedef struct card_struct {
    int card_value;
    struct card_struct *next_card;
}card_node;

card_node *front = NULL , *rear = NULL;
```

---

(4) Program functions:

`void enqueue(int card_in)`

> Put the cards into a linked list.

> The card at the front of the linked list is not the one to draw out, put it at the end of the linked list.

Usage

- `enqueue(card_in);`

Parameters

- `int card_in` : the value of card put back at the end of the linked list.

Return values

- no return values (void)

Code

```
void enqueue(int card_in)
{
    card_node *new_card_node = (card_node*)malloc(sizeof(card_node));
    new_card_node -> card_value = card_in;
    new_card_node->next_card = NULL;

    if (rear == NULL)
    {
        front  = new_card_node;
        rear = new_card_node;
    }
```

```
    else {
        rear->next_card = new_card_node;
        rear = new_card_node;
    }
    return;
}
```

`int dequeue(void)`

> draw out a card from the front of the linked list

Usage

- `card_out = dequeue();`

Return values

- `int card_out` : the value of the card which leaves from the queue.

Code

```
int dequeue(void)
{
    card_node *out_card_node = front;

    if (out_card_node != NULL) {
        front = front->next_card;
        int out_value = out_card_node->card_value;
        free(out_card_node);
        return out_value;
    }

    return -1;

}
```

---

(5) How I design my program :

How to draw out cards in order from K to A?

- I set `int index = 13` at begin, after draw out a card by `card_out = dequeue()` , using `if (card_out != index)` to check the value of the card is the one we want

(by order). If it is, do `index--` to change to the next target card value, otherwise put the card back by `enqueue(card_out)`.

How to deal with card values including numbers and capital letters?

- For doing `if (card_out != index)`, it needs to change `K Q J A` in to `13 12 11 1` by listing four cases, such as `if(!strcmp(input,"K")) card_in = 13;` . For string type `'2'~'10'` , use `card_in = atoi(input);` to converge to it's int type value.
- In the print out step, change `13 12 11 1` back to `K Q J A` by switch cases such as `if(ptr->card_value == 13) printf("K ");`.

How to print out all cards in the queue?

- Create another pointer `card_node *ptr = front`, use `ptr = ptr->next_card` move to the next card than print out the card value, finally stop when `ptr!= NULL`. My loop like this `for (card_node *ptr = front; ptr!= NULL; ptr = ptr->next_card)`.