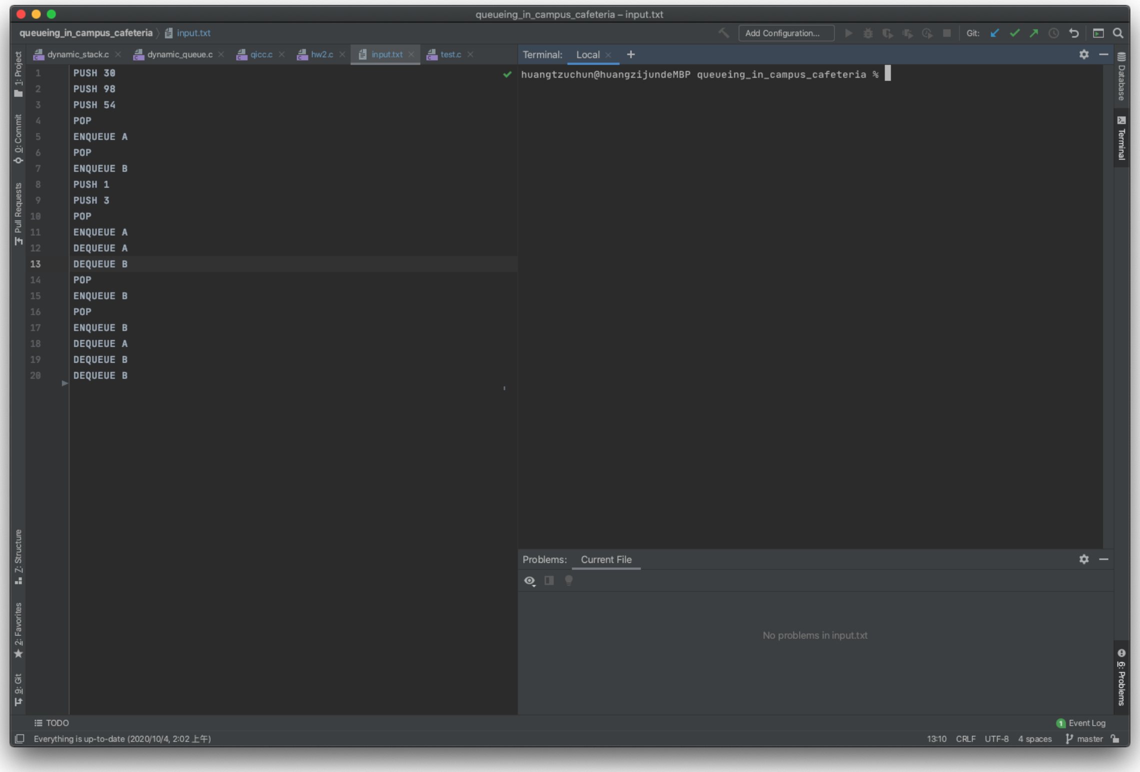
README

E14084117, CSIE112, 資訊二乙, 黃子峻

(1)Result screenshot:



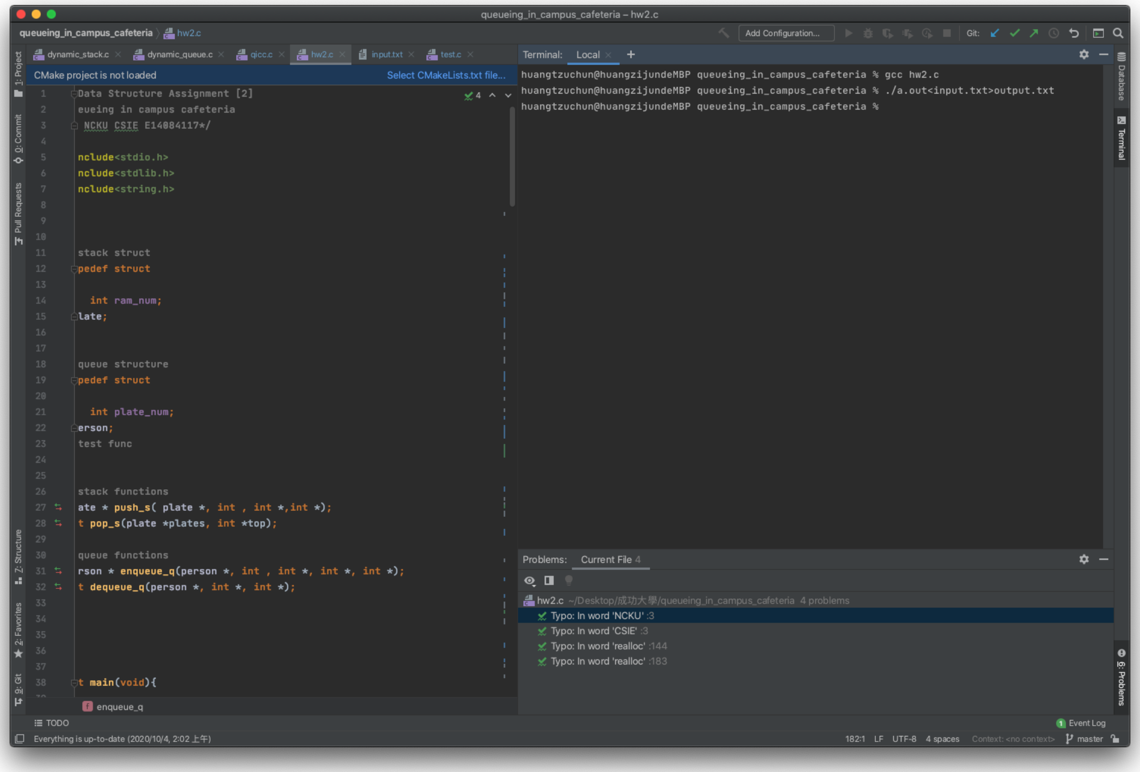
```
PUSH 30
PUSH 98
PUSH 54
POP
ENQUEUE A
POP
ENQUEUE B
PUSH 1
PUSH 3
POP
ENQUEUE A
DEQUEUE A
DEQUEUE B
POP
ENQUEUE B
POP
ENQUEUE B
DEQUEUE A
DEQUEUE B
DEQUEUE B
```

```
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria %
```

No problems in input.txt

Everything is up-to-date (2020/10/4, 2:02 上午)

13:10 CRLF UTF-8 4 spaces master



CMake project is not loaded   Select CMakeLists.txt file...

```
Data Structure Assignment [2]
eueing in campus cafeteria
 NCKU CSIE E14084117*/

nclude<stdio.h>
nclude<stdlib.h>
nclude<string.h>



stack struct
pedef struct

  int ram_num;
late;


queue structure
pedef struct

  int plate_num;
erson;
test func


stack functions
ate * push_s( plate *, int , int *,int *);
t pop_s(plate *plates, int *top);

queue functions
rson * enqueue_q(person *, int , int *, int *, int *);
t dequeue_q(person *, int *, int *);






t main(void){
```

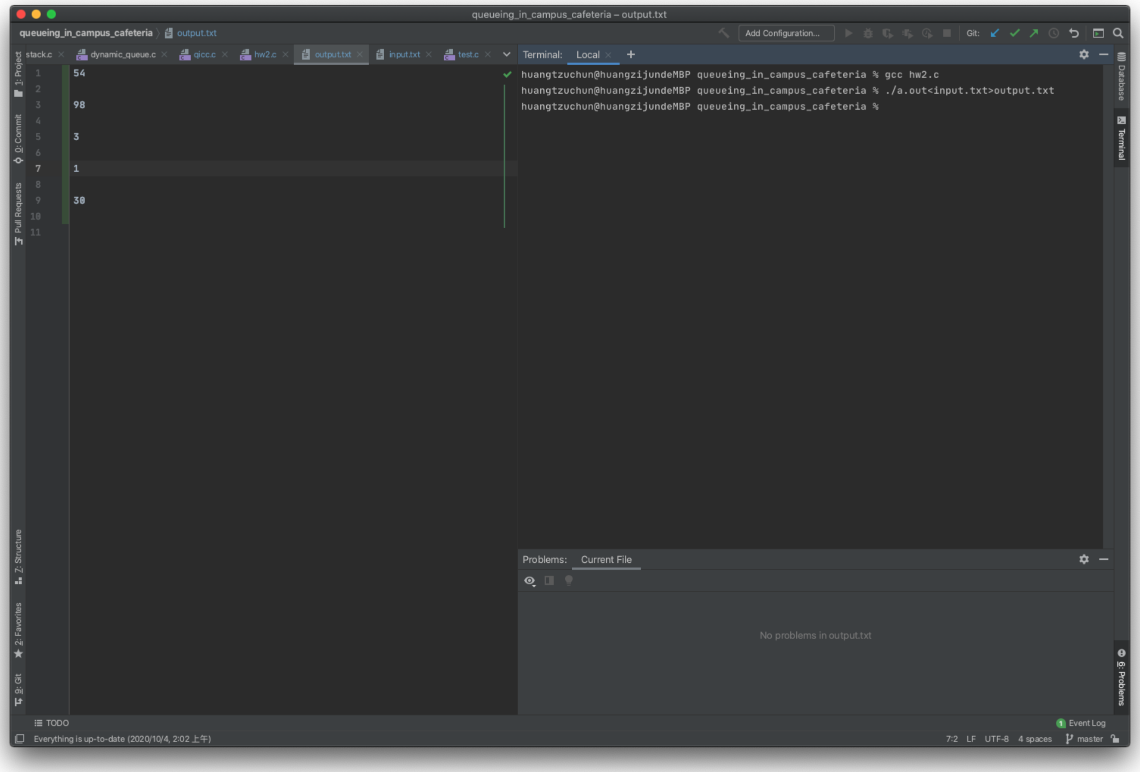
```
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria % gcc hw2.c
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria % ./a.out<input.txt>output.txt
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria %
```

Problems: Current File 4

hw2.c ~/Desktop/成功大學/queueing_in_campus_cafeteria 4 problems
- Typo: In word 'NCKU' :3
- Typo: In word 'CSIE' :3
- Typo: In word 'realloc' :144
- Typo: In word 'realloc' :183

Everything is up-to-date (2020/10/4, 2:02 上午)

182:1 LF UTF-8 4 spaces Context: <no context> master



```
54

98

3

1

30

```

```
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria % gcc hw2.c
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria % ./a.out<input.txt>output.txt
huangtzuchun@huangzijundeMBP queueing_in_campus_cafeteria %
```

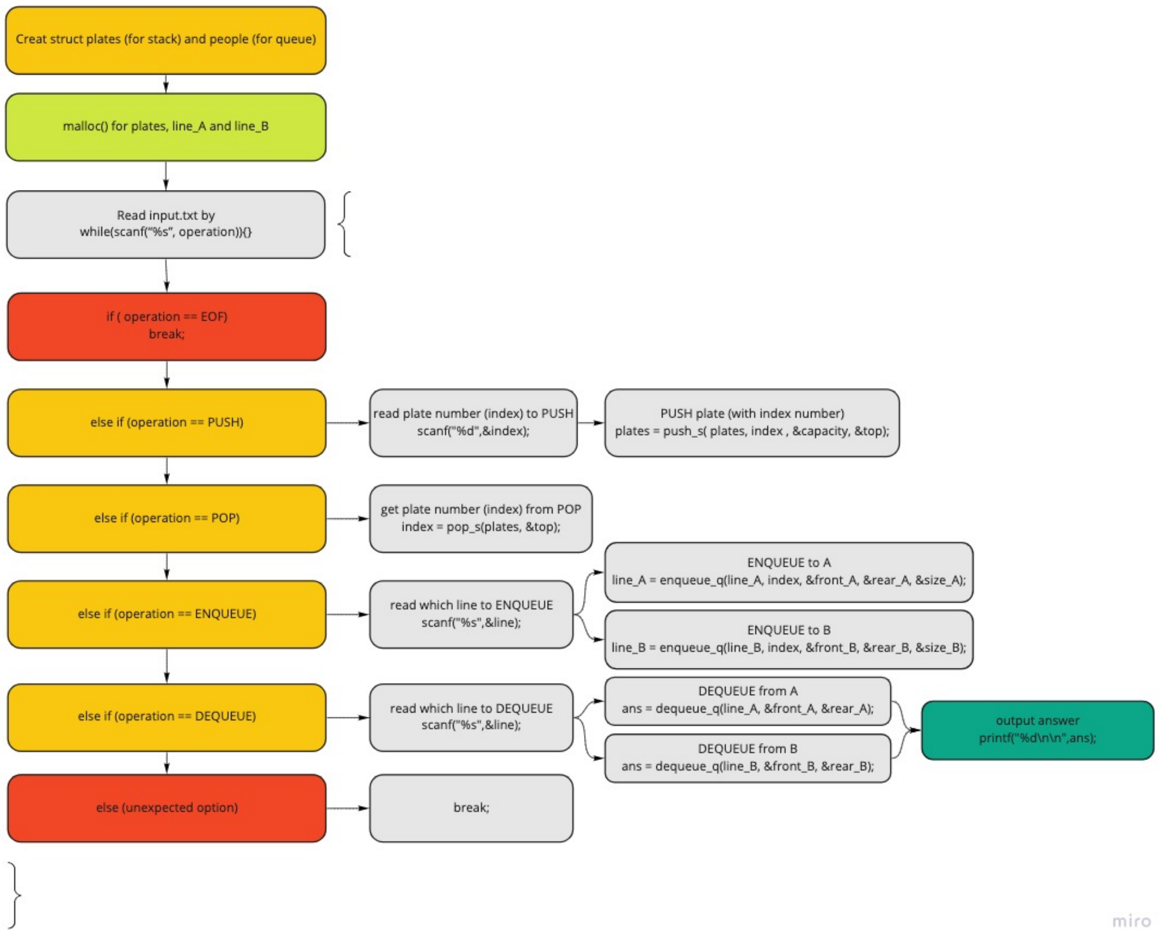
Problems: Current File

No problems in output.txt

Everything is up-to-date (2020/10/4, 2:02 上午)

7:2 LF UTF-8 4 spaces master

(2) Program architecture:


Creat struct plates (for stack) and people (for queue)
malloc() for plates, line_A and line_B
Read input.txt by while(scanf("%s", operation)){}
{
if (operation == EOF) break;
else if (operation == PUSH)
read plate number (index) to PUSH scanf("%d",&index);
PUSH plate (with index number) plates = push_s(plates, index , &capacity, &top);
else if (operation == POP)
get plate number (index) from POP index = pop_s(plates, &top);
else if (operation == ENQUEUE)
read which line to ENQUEUE scanf("%s",&line);
ENQUEUE to A line_A = enqueue_q(line_A, index, &front_A, &rear_A, &size_A);
ENQUEUE to B line_B = enqueue_q(line_B, index, &front_B, &rear_B, &size_B);
else if (operation == DEQUEUE)
read which line to DEQUEUE scanf("%s",&line);
DEQUEUE from A ans = dequeue_q(line_A, &front_A, &rear_A);
DEQUEUE from B ans = dequeue_q(line_B, &front_B, &rear_B);
output answer printf("%d\n\n",ans);
else (unexpected option)
break;
}
miro

(3)Program structs:

Struct for stack

```
typedef struct
{
    int ram_num;
}plate;
```

- `ram_num` is for store the random number, which is assigned to the plate.

Struct for Queue

```
typedef struct
{
    int plate_num;
}person;
```

- `plate_num` is the same idea as `ran_num` mention above.

---

(4) Program functions:

`plate * push_s( plate *plates, int index, int *capacity,int *top)`

> indicates the staff refills a plate index N into the plate stack.

Usage

- `plates = push_s( plates, index , &capacity, &top);`

Parameters

- `plate *plates` : `plate` type pointer which points to the `plate` type dynamic storage with stack property.
- `int index` : the random number of plate want to store in stack.
- `int *capacity` : this pointer pointing to `&capacity`, for recording the capacity of the stack.
- `int *top` : this pointer pointing to `&top`, the index which locates the last element in the stack.

Return values

- `plate *` : `plate` type pointer which points to the `plate` type dynamic storage. It's for the realloc() process.

`int pop_s(plate *plates, int *top)`

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

Usage

- `index = pop_s(plates, &top);`

Parameters

- `plate *plates` : `plate` type pointer which points to the `plate` type dynamic storage with stack property.
- `int *top` : this pointer pointing to `&top`, the index which locates the last element in the stack.

Return values

- `int` type plate number which is removed from `plates` stack.

`person * enqueue_q(person *line_choose, int index, int *front, int *rear, int *line_size)`

> indicates a customer joins the end of the line X.

Usage

- `line_A = enqueue_q(line_A, index, &front_A, &rear_A, &size_A);`
- `line_B = enqueue_q(line_B, index, &front_B, &rear_B, &size_B);`

Parameters

- `person *line_choose` : `person` type pointer which pointing to the `person` type dynamic storage with queue property.
- `int index` : the plate number we want to enqueue to the `*line_choose` queue, which gets from `pop_s()`.
- `int *front` / `int *rear` : `*front` point to `&front` , `front` is the index number of front in queue; `*rear` point to `&rear` , `rear` is the index number of rear in queue.
- `int *line_size` : `*line_size` point to `&line_size`, `line_size` is the capacity of queue.

Return values

- `person` type pointer, point to the memory of queue. It's for the realloc() process.

`int dequeue_q(person *line_choose, int *front, int *rear)`

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

Usage

- `ans = dequeue_q(line_A, &front_A, &rear_A);`
- `ans = dequeue_q(line_B, &front_B, &rear_B);`

Parameters

- `person *line_choose` : `person` type pointer which pointing to the `person` type dynamic storage with queue property.
- `int *front` / `int *rear` : `*front` point to `&front` , `front` is the index number of front in queue; `*rear` point to `&rear` , `rear` is the index number of rear in queue.

Return values

- `int` type plate number which leaves from the line (queue).

---

(5) How I design my program :

How I build the plate stack?

- By using `top` as an index number of the last element in the stack, we can `PUSH` and `POP` data by running `push_s()` and `pop_s()` functions which operate to `plates[top].ram_num`.
- Considering the real-world case, the tower of plates has a limit high. In case of an unknown amount of plates in the stack, I choose to `alloc()` 10 size in the beginning, and `relloc()` 10 size a time when it reaches the memory limit.
- In the PUSH process, my `push_s()` function will first use `top == capacity-1`(-1 because `top` start by -1 to let `top = 0` when stack has one element) to check whether the stack is full or not. If stack is full, the function will do `relloc()` to append more memory for stack.

How I build the line queue?

- By using `front` and `rear` as an index number of the first and last element in queue, we can `ENQUEUE` data to `line_choose[rear].plate_num` by running `enqueue_q()` function; and `DEQUEUE` data from `line_choose[front].plate_num` by running `dequeue_q()` function.
- Considering the real-world case, the line of people in the cafeteria has a limit length. In case of an unknown amount of people in the line, I choose to `alloc()` 10 size in the beginning, and `relloc()` 10 size a time when it reaches the memory limit.
- In the ENQUEUE process, my `enqueue_q()` function will first use `rear == line_size-1` (-1 because `rear` start by -1 to let `rear= 0` when queue has first element) to check whether queue is full or not. If queue is full, the function will do `relloc()` to append more memory for queue.
- After we have done `dequeue_q()` few times, there will have some unused memory in front of `line_choose[front]`, I didn't choose to `relloc()` memory to free it because of it may take a while to assign each element to the different memory location and we assume there won't be too many data(people in the cafeteria has limited amount) .

Why I define the struct for stack and queue instead of using int dynamic allocate int array?

- If one day we have more than one type of data on a plate (e.g.: red color plate with number 13) that needs to be considered, it easy to change the struct configuration for the new requirement.

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`.