**EECS2040 Data Structure Hw #2 (Chapter 3 Stack/Queue)**

**due date 4/17/2022**

**by 108011234 陳昭維**

**Part 2 Coding (5% of final Grade)**

You should submit:

(a) All your source codes (C++ file). Proper comments should be included in the code.

(b) Show the execution trace of your program.

1. (30%) Based on the circular queue and template queue ADT in **ADT 3.2** shown below (or pptx pp.82), write a C++ program to implement the queue ADT using dynamic (circular) array (10%)**.** Then add three more functions to
2. (5%) Return the size of a queue (int Size()).
3. (5%) Return the capacity of a queue (int Capacity()).
4. (10%) Merge two queues into a one by alternately taking elements from each queue. The relative order of queue elements is unchanged. What is the complexity of your function?

You should **demonstrate all the functions** using at least one example.

**template** < **class** T >

**class** Queue

{

**public**:

Queue (**int** queueCapacity = 0);

**bool** IsEmpty( ) **const**;

**void** Push(**const** T& item); // add an item into the queue

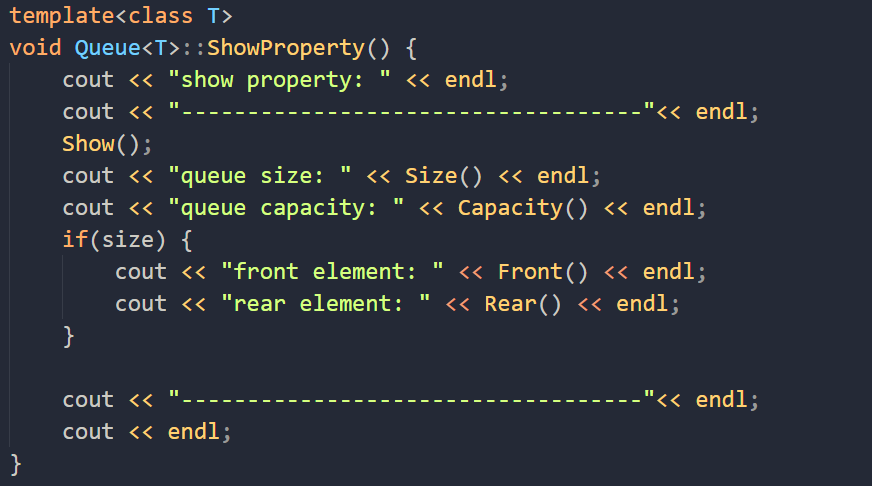
**void** Pop( ); // delete an item

T& Front() const; // return top element of stack

T& Rear() const; // return top element of stack

} ;

**<Answer>**

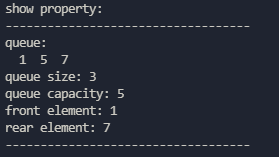
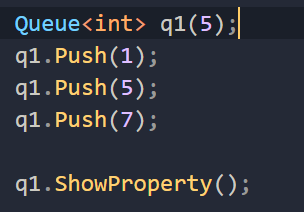
****

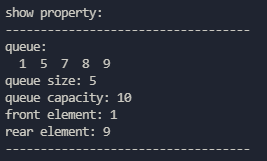
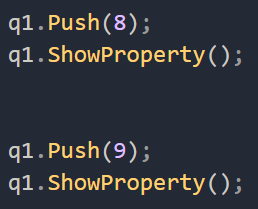
Demo 會多次利用到Queue 的class method, showProperty() 來展示queue的變化和內容。

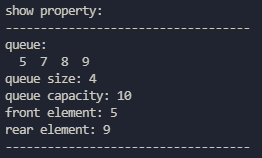
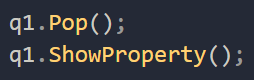
其中也包含了 **part(a) 的 int Size()**

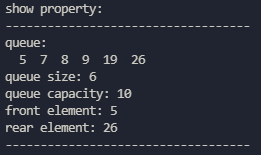
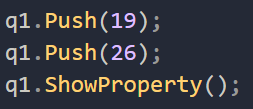
以及 **part(b) 的 int Capacity()**

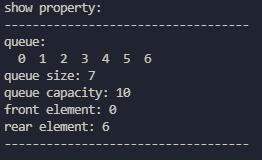
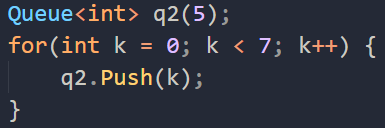
以及我自己定義的Show() function 來展示queue 的元素，以及使用Front()和Rear來看 queue最前和最後的元素。

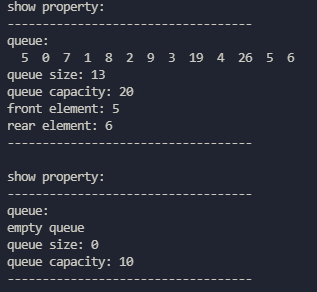
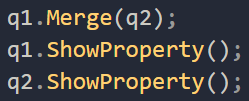
** **

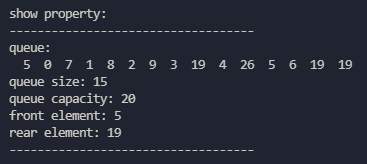
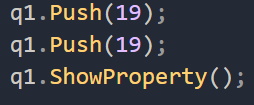
** **

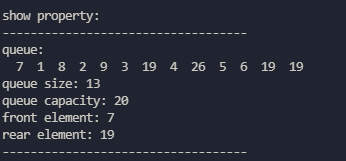
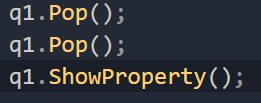
** **

** **

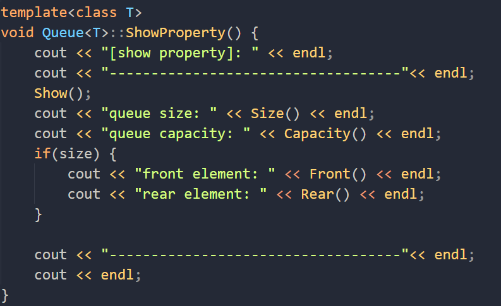
 

**\**

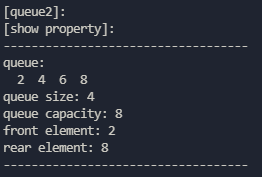
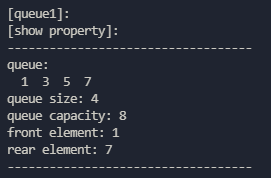
The time complexity of merge is O(size of queue1 + size of queue2)

1. (20%) Design a C++ function template, reverseQueue, that takes as a parameter a queue object and uses a stack object to reverse the elements of the queue. The operations on queue and stack should strictly follow the ADT 3.2 Queue ADT and ADT 3.1 Stack ADT. You should **demonstrate the functions** using at least one example, e.g., queue1=(1,3,5,7), queue2=(2,4,6,8), mergedqueue=(1,2,3,4,5,6,7,8)

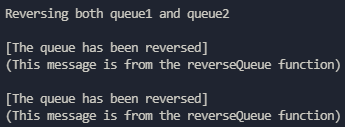
****

****

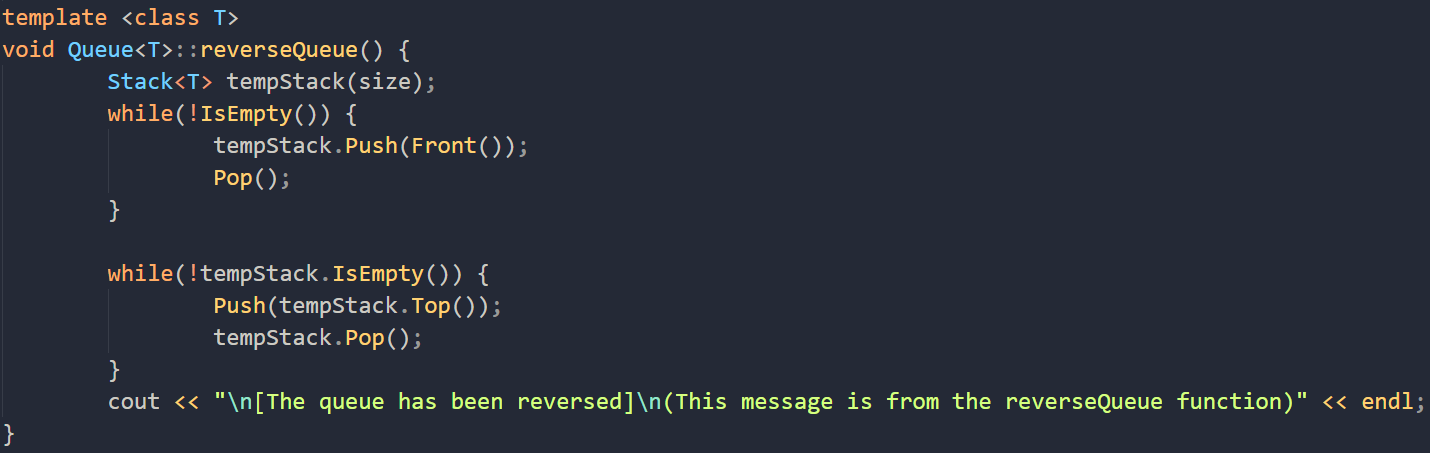
Demo 會多次利用到Queue 的class method, showProperty() 來展示queue的變化和內容

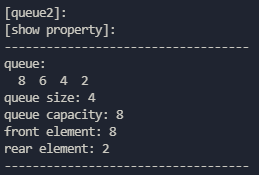
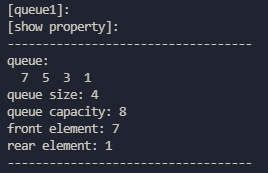
****

先用showProperty() 展示queue1 和 queue2的內容

****

再利用class method, reverseQueue() 來反轉整個queue

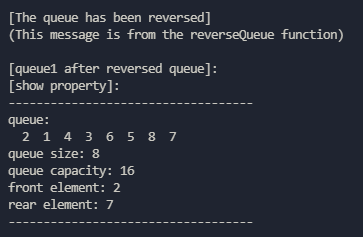
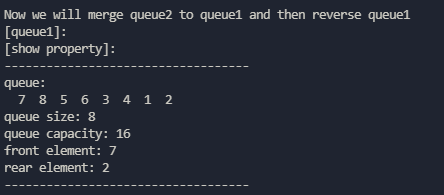
****

****

再來我們使用第一題的merge function 把queue2 merge到 queue1. 再來再使用reverseQueue() 來反轉整個merged過的 queue1

Merge過的應該會長(7,8,5,6,3,4,1,2)

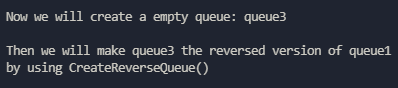
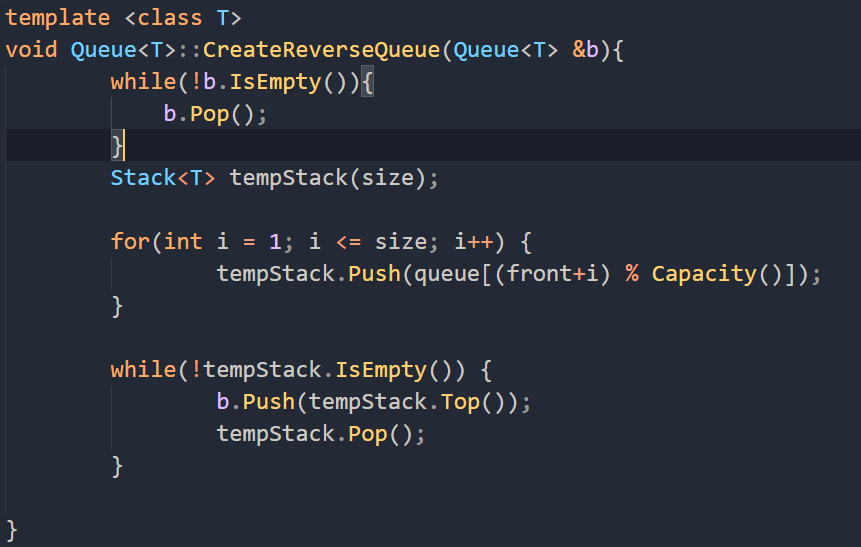
Reverse過會長(2,1,4,3,6,5,8,7)

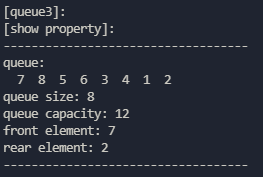
****

最終我們再來使用另一個class method, CreateReverseQueue() 來把一個queue object便成一個queue的反轉版本

所以原先empty queue3會變成 queue1的倒轉版本

(7,8,5,6,3,4,1,2)

****

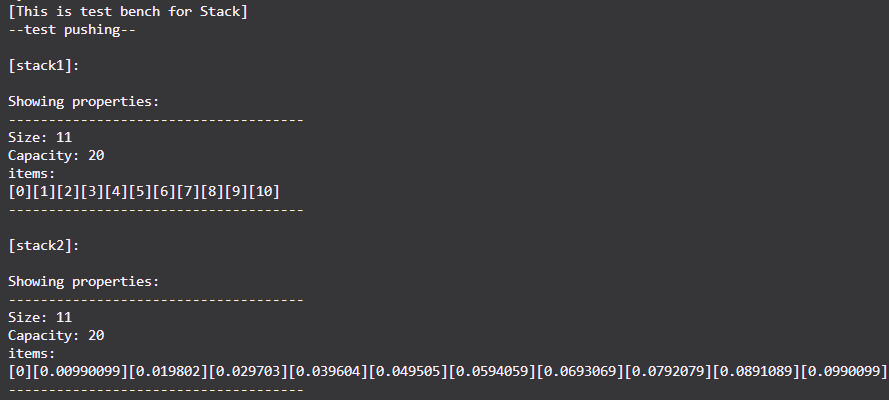
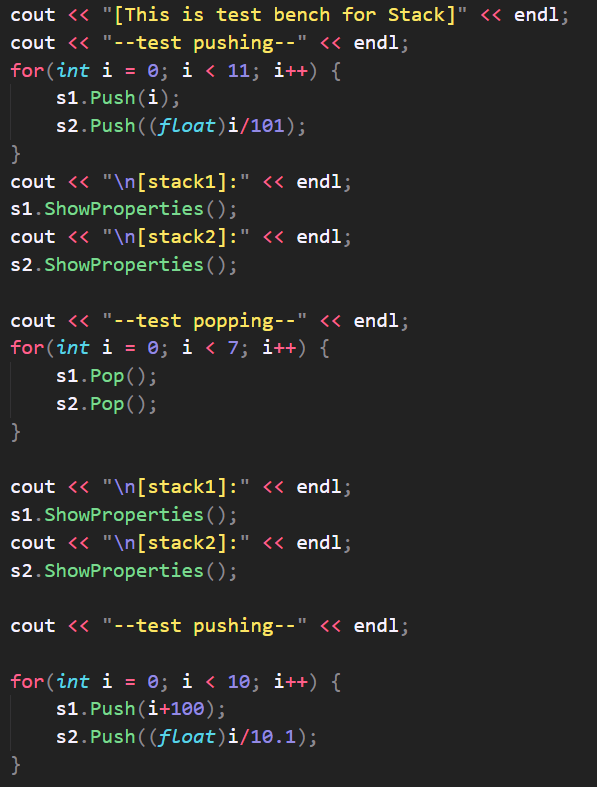
****

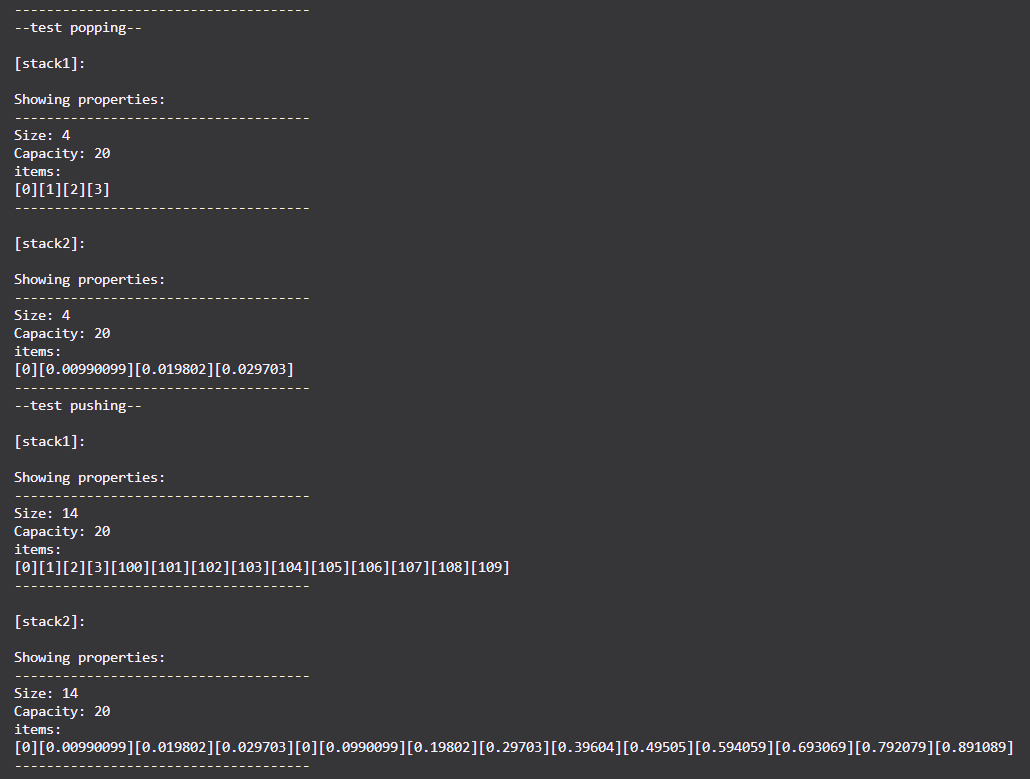
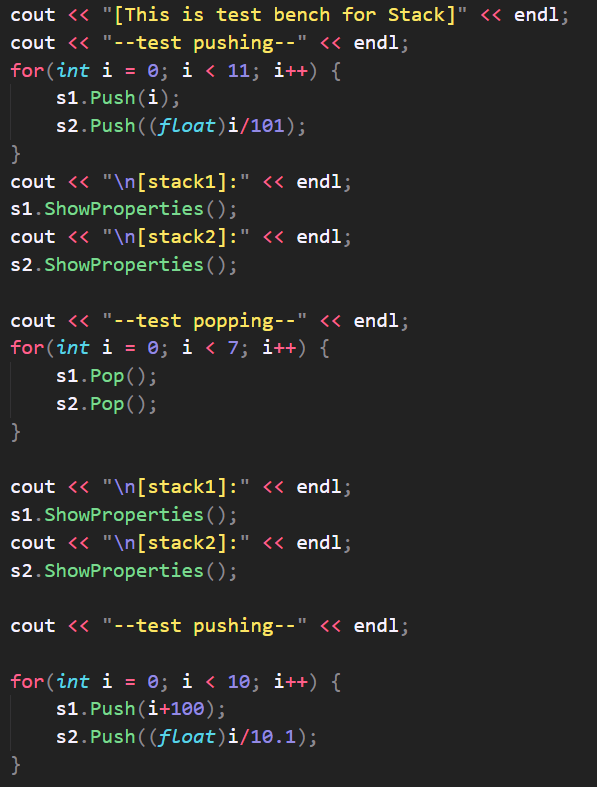
1. (25%) Referring to **Program 3.13** in textbook (pptx pp.98),
2. (5%) Implement Stack as a publicly derived class of Bag using template. **Demonstrate** your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of Pushes and Pops and Size functions.
3. (5%) Implement Queue as a publicly derived class of Bag using template. **Demonstrate** your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of Pushes and Pops and Size functions.
4. (15%) A template double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Implement the class Deque as a publicly derived templated class of Queue. The class Deque must have public functions (either via inheritance from Queue or by direct implementation in Deque) to add and delete elements from either end of the deque and also to return an element from either end. The complexity of each function (excluding array doubling) should be (1).

**Demonstrate** your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of two types of Pushes and Pops and Size functions to illustrate your code is working.

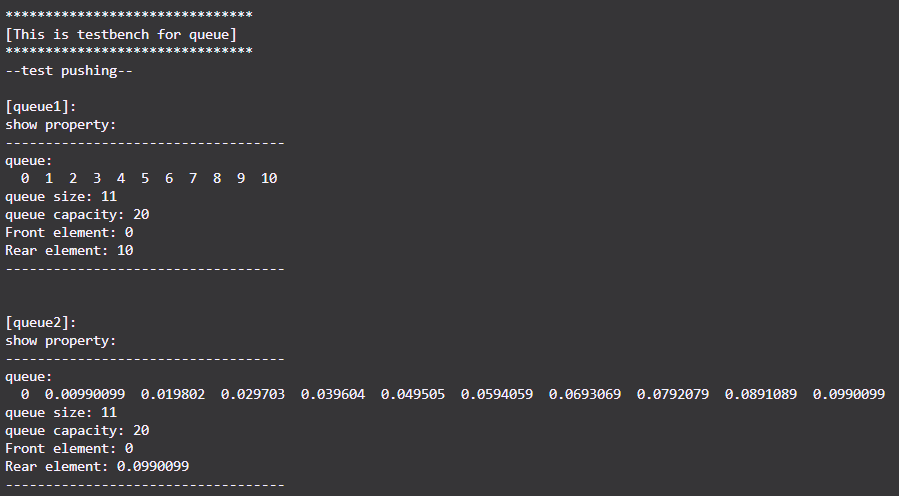
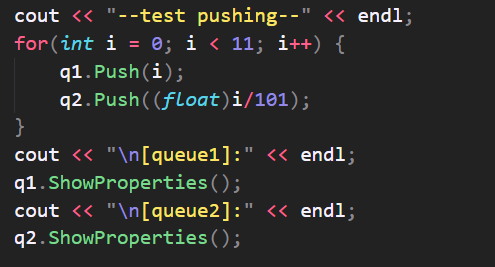
**<answer>**

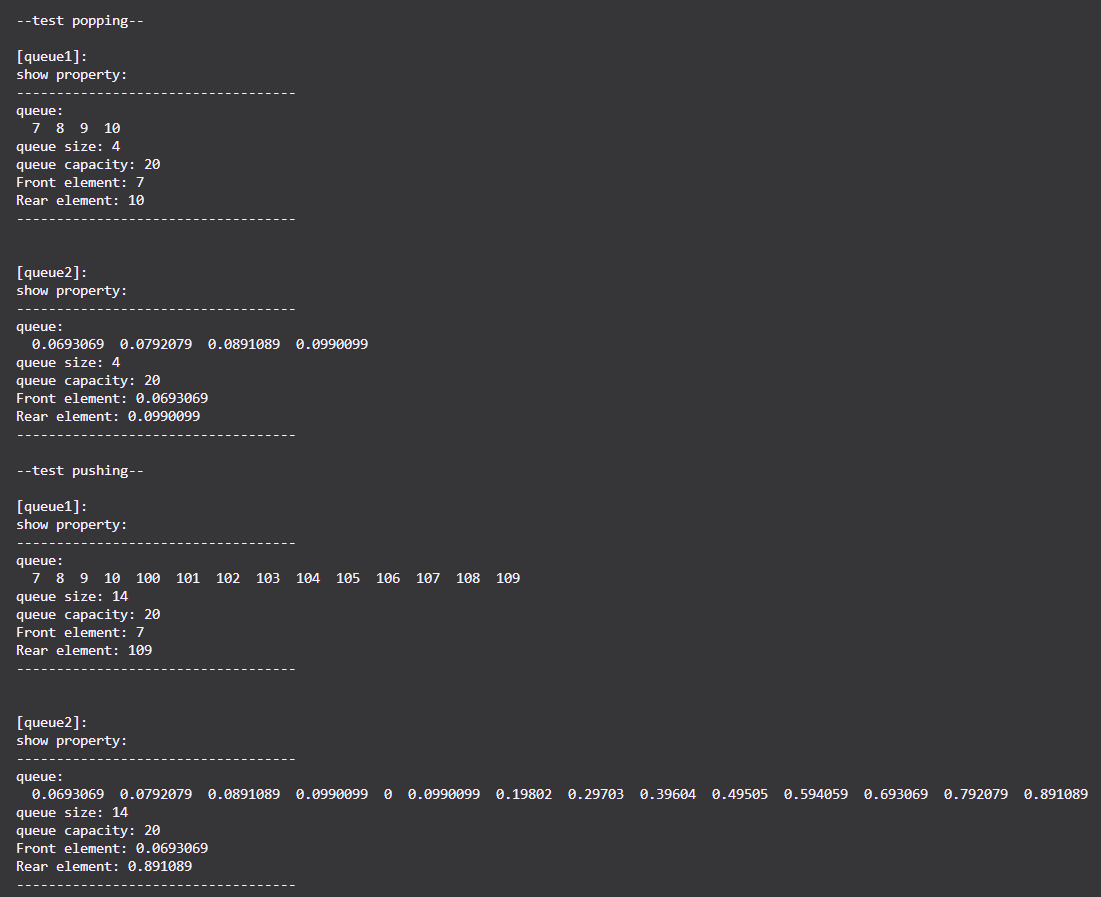
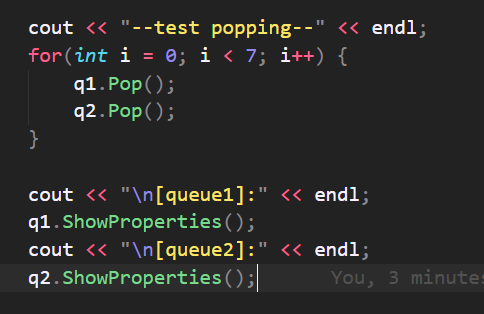
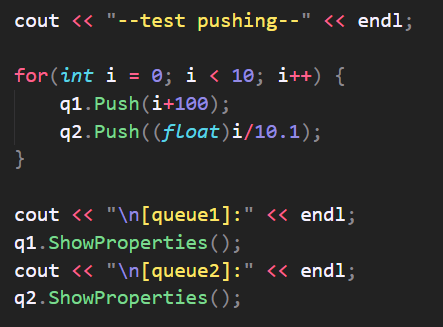
**Test for stack**

****

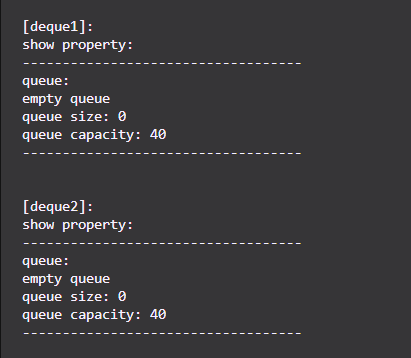
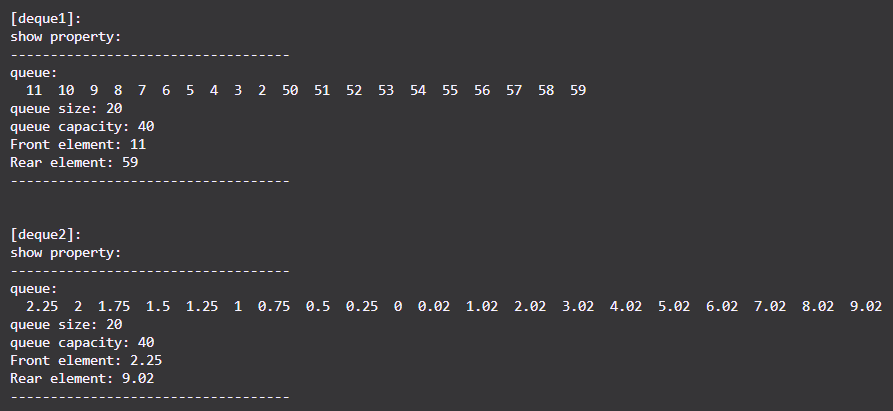
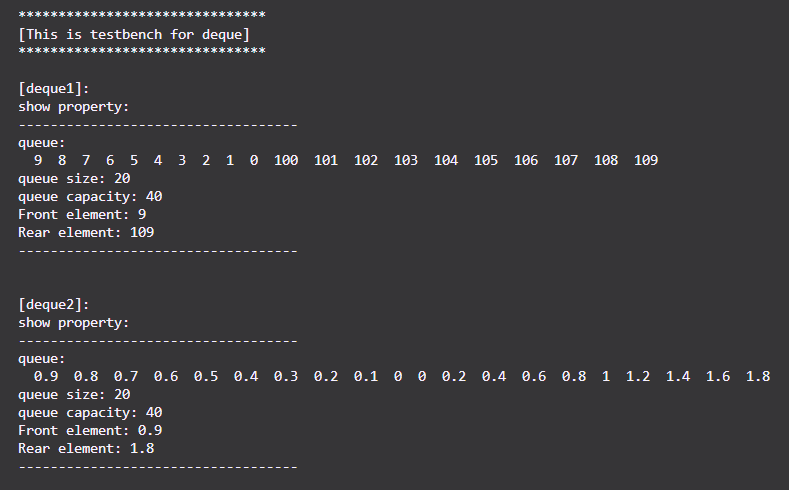
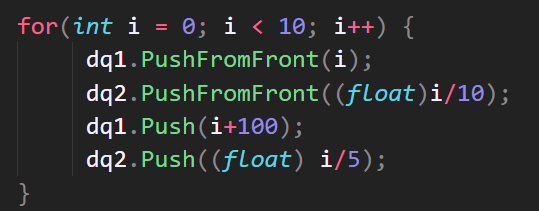
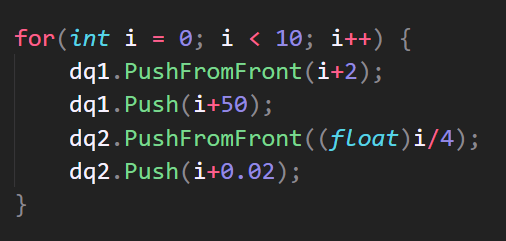
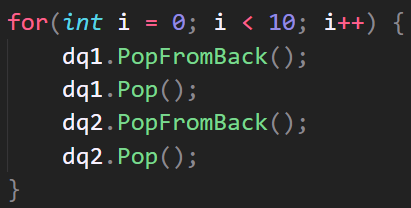
****

**Test for queue**

****

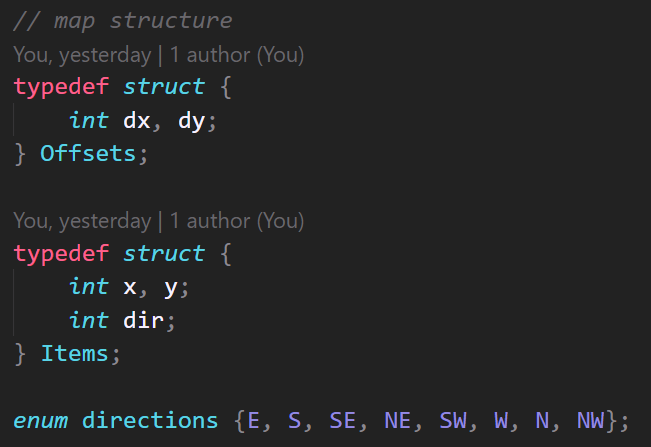
****

**Test for deque**

**** (25%) Write a C++ program to implement the maze in textbook using the example codes of **Program 3.15** (pptx pp.106 Algorithm()) and **3.16 (pptx void Path(const int m, const int p)**. You should use a text editor to edit a file containing the maze matrix and then **read in the file** **to establish the maze matrix** in your program. The default entrance and exit are located in the upper left corner and lower right corner, respectively as shown in textbook.

1. (10%) Demonstrate your maze program using the maze shown in **Figure 3.11**.
2. (5%) Find a path manually through the maze shown in **Figure 3.11**.
3. (10%) Trace out the action of function path (**Program 3.16**) on the maze shown in Figure 3.11. Compare this to your own attempt in (b).

<answer>

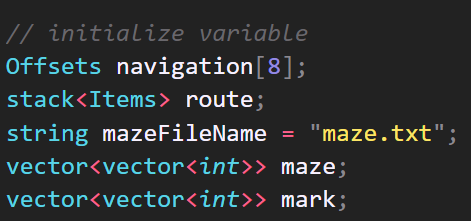


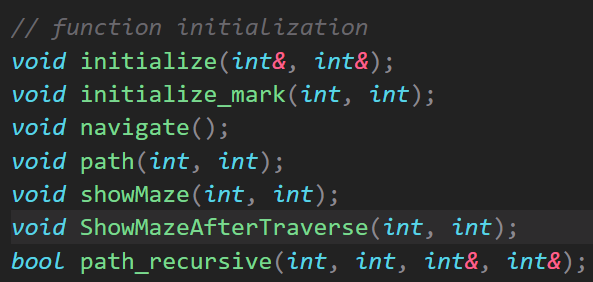
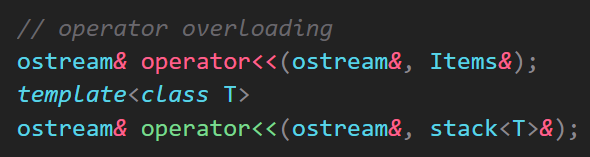
Initialize some of the basic variables for the map trace, Offsets struct for the navigation direction, we will create an Offsets array for different (y,x) tuple to move.

Then create Items structure for later stack objects that stores direction and current position.

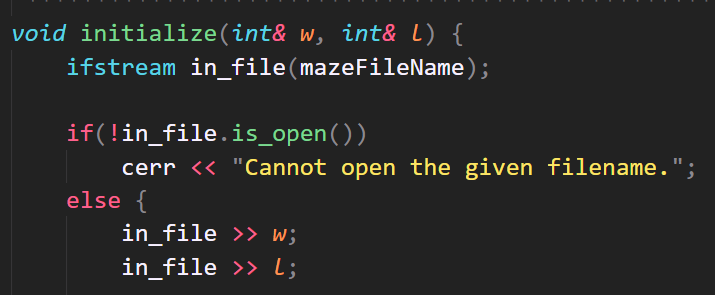
And then initialize the maze file name as string.

Initialize a vector of vector<int> to store the map element. Also initialize a identical size map to mark the visited grid.



****

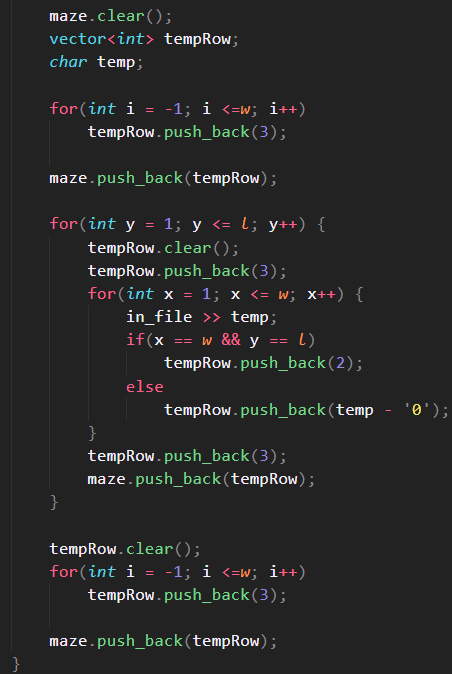
Some function initializations

****

Read file from given file name.

Output exceptions if file not found.

Read the width and length of the map in the first row of the text file

****

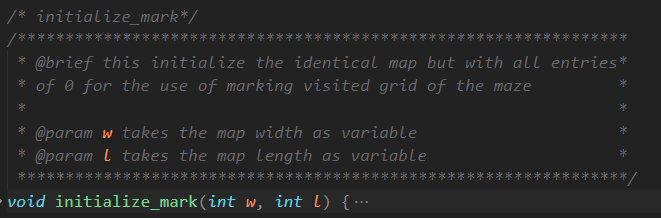
Read file from given file name.

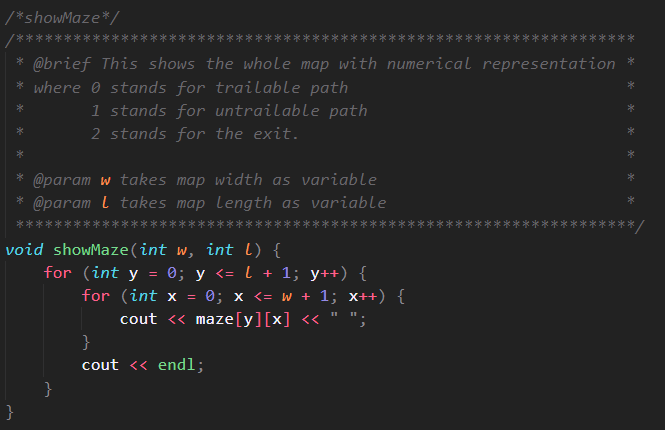
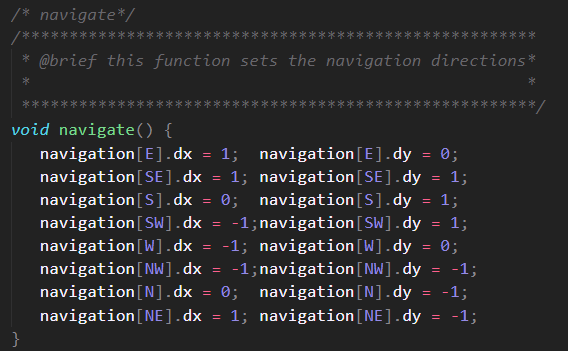
Output exceptions if file not found.

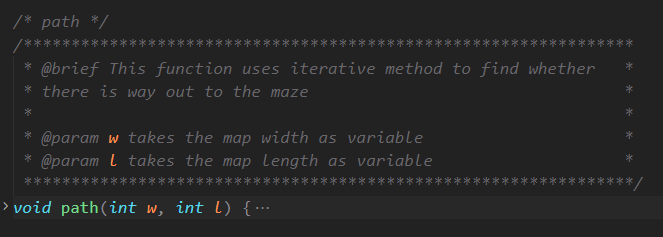
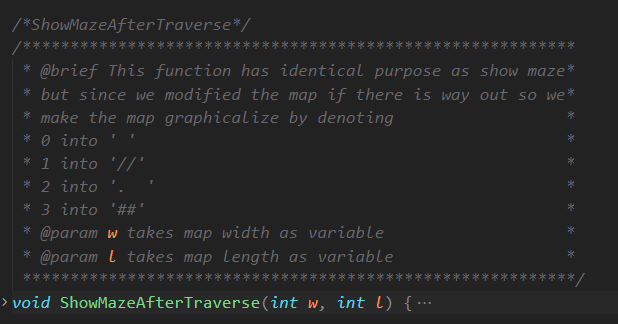
Read the width and length of the map in the first row of the text file

Then sequentially read every grid data and push it into a tempRow vector, and after one row, the program pushes the vector to the map (vector of vector).

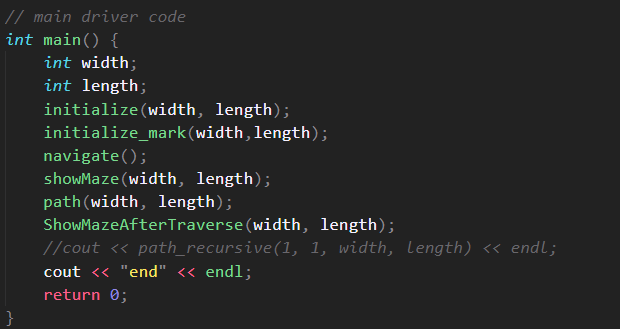
We also pushes boundary grid which is consists of integer 3.

****

****

****

**Driver code:**

****

First, we initialize width and length variables, then we initialize the map and the mark vector of vectors.

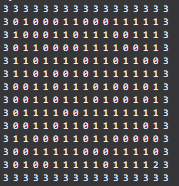
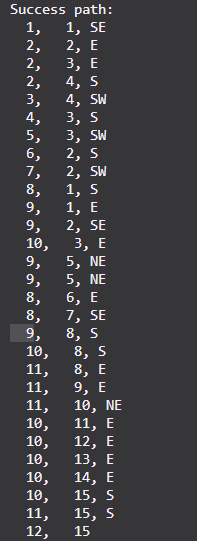
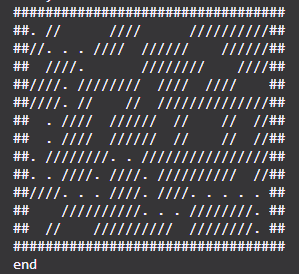
Then we initialize the navigation direction.

And show the raw map before traversing.

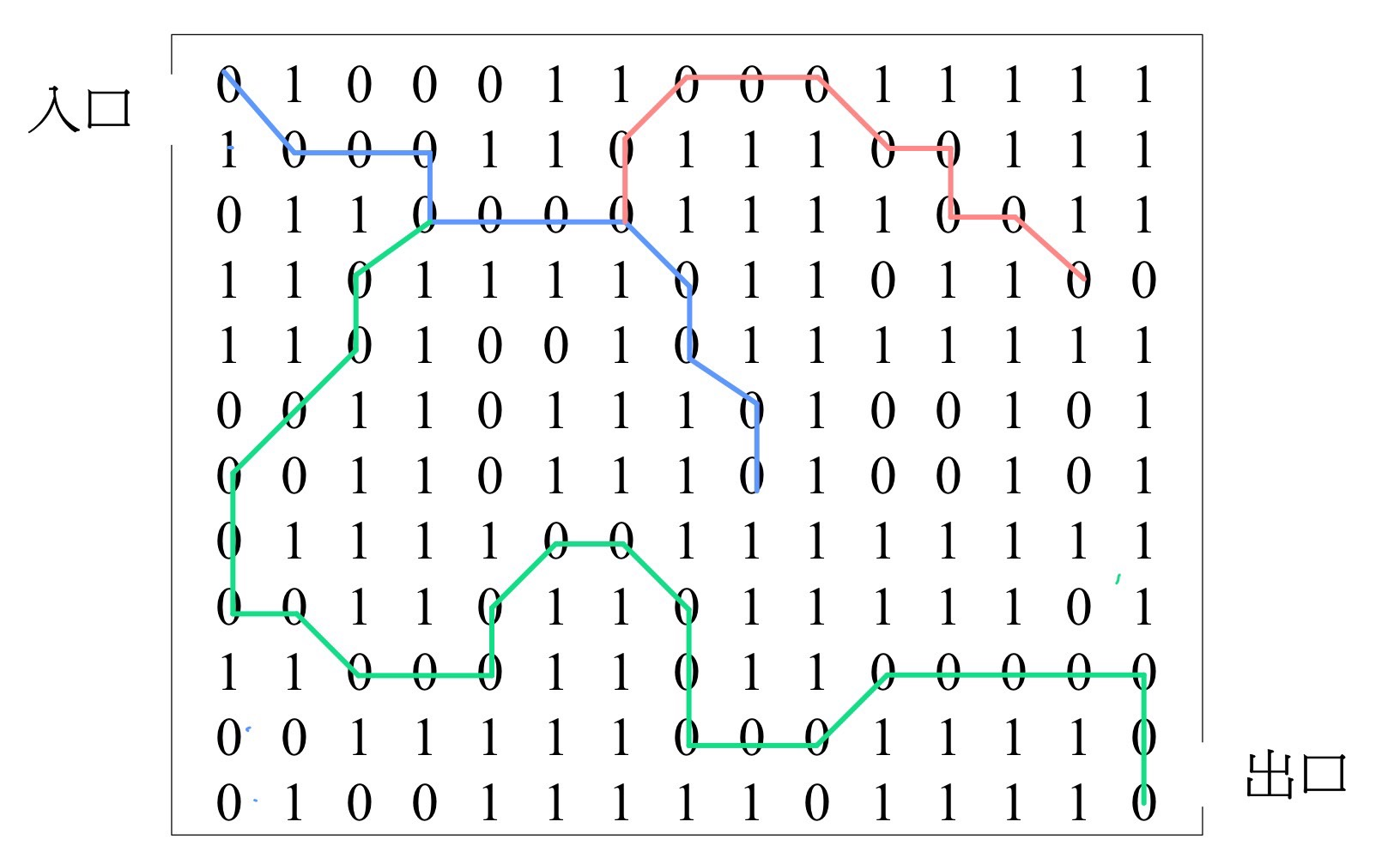
Use the path function to find the path out and list out the path

Then show the map after traversing.

**(a)**

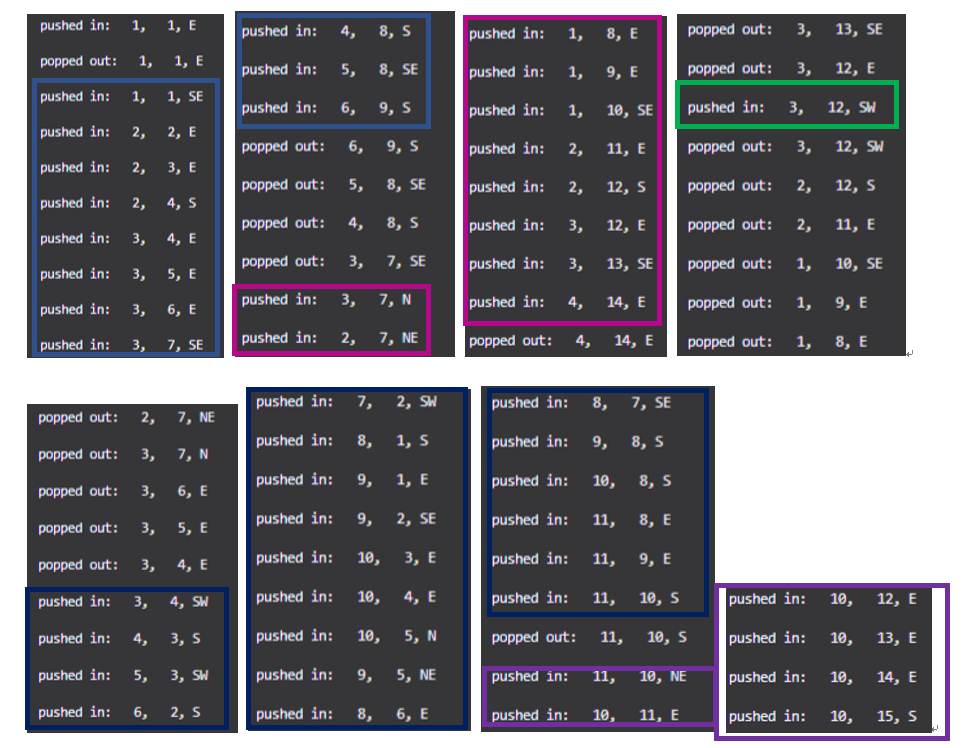
****  **** ****

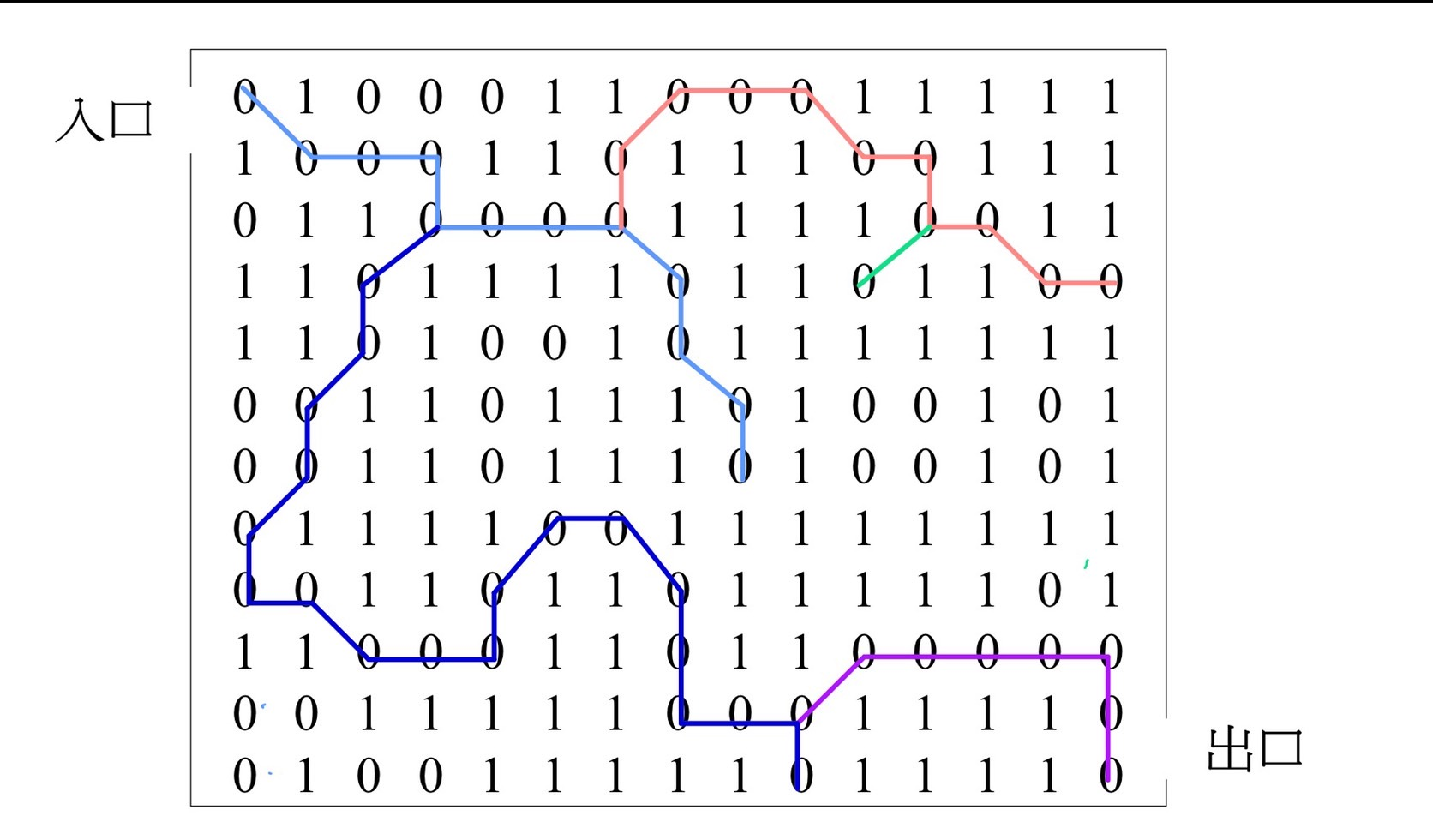
**(b)**



My attempt has 2 failure before finding the path out.

**(c)**

****



The attempt of the path function has 4 failure before finding way out. Human attempts has less failure because we can scrutinize the whole map and gaining insights of what to move for further step, but the Path() uses a greedy algorithm DFS to search the way out of the current state, thus it takes a lot more step.