EE2410 Data Structure Coding HW #2 (Chapter 3 of textbook)

due date 4/14/2024(Sun.) 23:59

Student No.: ____111060005_____

Name: _____胡昱煊_____

You should submit:

(a) All your source codes (C++ file).

(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data.

Submit your homework before the deadline (midnight of 4/14). Fail to comply (**late** homework) will have ZERO score. **Copy** homework will have SERIOUS consequences.

Stacks & Queues: **due date: 23:59, 4/14/2024 (Sun.)**

1. (30%)

Referring to **Program 3.13** (definition of Bag and Stack):

```
Class Bag
{
public:
    Bag (int bagCapacity = 10);
    virtual  ~Bag( );
    virtual int Size( ) const;
    virtual bool IsEmpty( ) const;
    virtual int Element( ) const;
    virtual void Push(const int);
    virtual void Pop( );
protected:
    int *array;
    int top;
};
```

```
class Stack : public Bag
{
public:
    Stack (int stackCapacity = 10);
    ~Stack( );
    int Top( ) const;
    void Pop( );
};
```

(a) (15%) 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.

(b) (15%) 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.

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.

(a)

D:\C++\data_structure\hw2\q1_a.exe

```
type in some int
How many elements do you want to push? 6
1 2 3 4 5 6
S1 = Stack: ( 1, 2, 3, 4, 5, 6 )
The size of S1 is : 6
S1 after pop = Stack: ( 1, 2, 3, 4, 5 )
What element do you wan't to put into S1? 8
S1 = Stack: ( 1, 2, 3, 4, 5, 8 )

type in some float
How many elements do you want to push? 5
0.1 0.2 0.3 0.4 0.5
S2 = Stack: ( 0.1, 0.2, 0.3, 0.4, 0.5 )
The size of S2 is : 5
S2 after pop = Stack: ( 0.1, 0.2, 0.3, 0.4 )
What element do you wan't to put into S2? 0.8
S2 = Stack: ( 0.1, 0.2, 0.3, 0.4, 0.8 )

Process returned 0 (0x0)   execution time : 64.312 s
Press any key to continue.
```

(b)

D:\C++\data_structure\hw2\q1_b.exe

```
type in some int
How many elements do you want to push? 6
1 2 3 4 5 6
S1 = Queue: ( 1, 2, 3, 4, 5, 6 )
The size of S1 is : 6
S1 after pop = Queue: ( 2, 3, 4, 5, 6 )
What element do you wan't to put in S1? 8
S1 = Queue: ( 2, 3, 4, 5, 6, 8 )

type in some float
How many elements do you want to push? 5
0.1 0.2 0.3 0.4 0.5
S2 = Queue: ( 0.1, 0.2, 0.3, 0.4, 0.5 )
The size of S2 is : 5
S2 after pop = Queue: ( 0.2, 0.3, 0.4, 0.5 )
What element do you wan't to put in S2? 0.8
S2 = Queue: ( 0.2, 0.3, 0.4, 0.5, 0.8 )

Process returned 0 (0x0)   execution time : 23.436 s
Press any key to continue.
```

2. (35%)

Based on the circular queue and template queue ADT in **ADT 3.2** shown below, write a C++ program to implement the queue ADT using dynamic (circular) array**.** Then add following functions to

(a) (5%) Return the size of a queue (int Size()).

(b) (5%) Return the capacity of a queue (int Capacity()).

(c) (5%) Overload the relational operator == for the class Queue that returns true if two queues of the same type are the same, false otherwise. (Two queues of the same type are the same if they have the same number of elements and their elements at the corresponding positions are the same.)

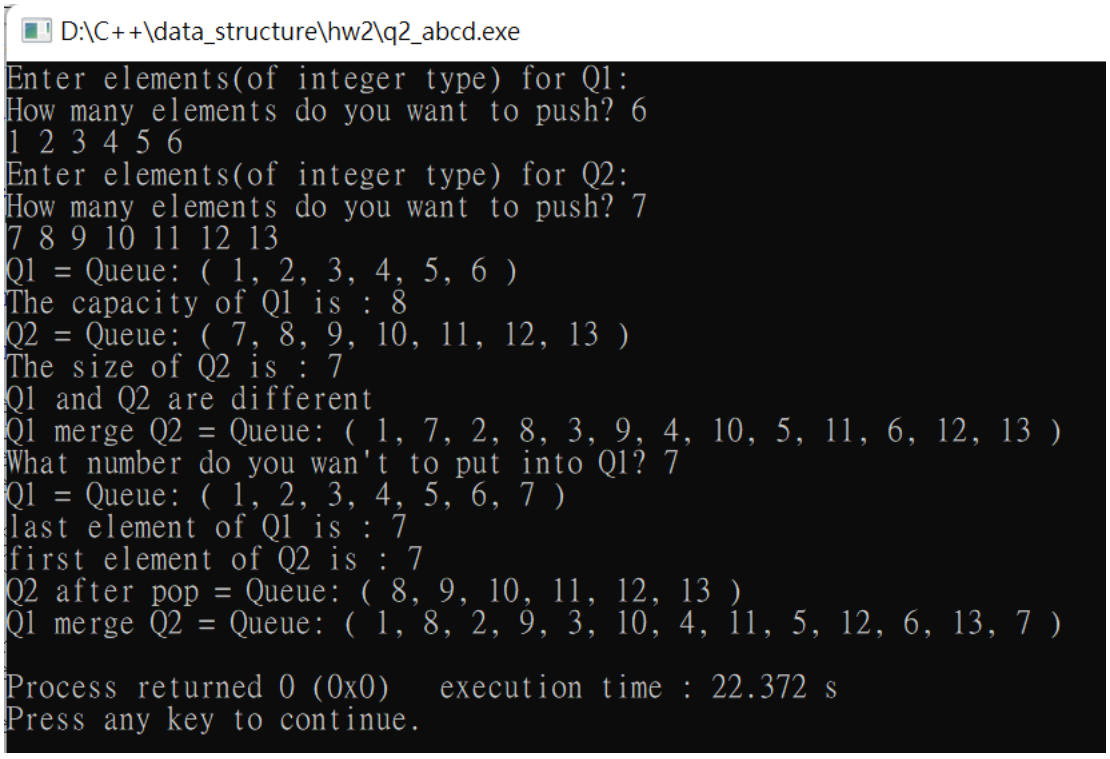
(d) (10%) Merge two queues into one by alternately taking elements from each queue. The relative order of queue elements is unchanged. What is the complexity of your function? You should **demonstrate the functions** using at least one example, e.g., queue1=(1,3,5,7), queue2=(2,4,6,8), merged queue=(1,2,3,4,5,6,7,8)

(e) (10%) Reverse the queue (ReverseQueue()), that uses a stack object to reverse the elements of the queue.
You should **demonstrate the functions** using at least one example, e.g., queue1=(1,3,5,7), stack = (), after reverse, queue1=(7,5,3,1).

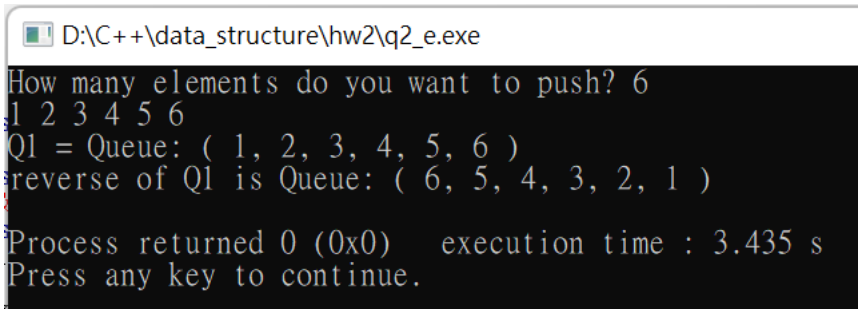
You should **demonstrate all the functions** using at least one example.

```
template < class T >
class Queue
{
public:
    Queue (int queueCapacity = 0);
    ~Queue();
    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
private:
    //omitted
} ;
```

(a)(b)(c)(d)

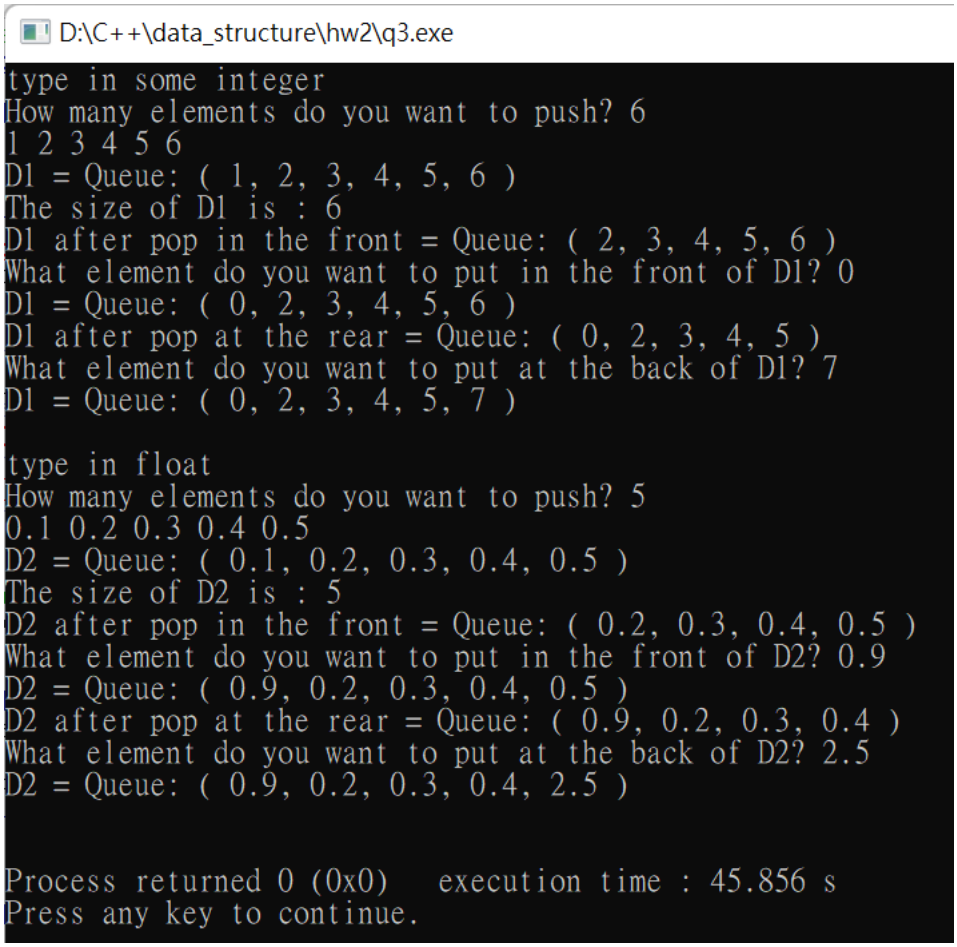


(e)



3. (20%)

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 (using circular array). 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 (add PushFront() and PopRear() functions) and also to return an element from either end. The complexity of each function (excluding array doubling) should be $\Theta(1)$.



4. (15%)

Write a C++ program to implement the maze in textbook using the example codes of **Program 3.15** and **3.16**. 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.

(a) (5%) Demonstrate your maze program using the maze shown in **Figure 3.11**.

(b) (5%) Find a path manually through the maze shown in **Figure 3.11**.

(c) (5%) 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).

入口

0	1	0	0	0	1	1	0	0	0	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	0	0	1	1	1
0	1	1	0	0	0	0	1	1	1	1	0	0	1	1
1	1	0	1	1	1	1	0	1	1	0	1	1	0	0
1	1	0	1	0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	0	1	0	0	1	0	1
0	0	1	1	0	1	1	1	0	1	0	0	1	0	1
0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	0	1	1	1	1	1	0	1
1	1	0	0	0	1	1	0	1	1	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0	1	1	1	1	0
0	1	0	0	1	1	1	1	1	0	1	1	1	1	0

出口

Figure 3.11：一個迷宮的例子（你能找出一條路徑嗎？）

(a)

```
D:\C++\data_structure\hw2\q4.exe
The maze is :
0 1 0 0 0 1 1 0 0 0 1 1 1 1 1
1 0 0 0 1 1 0 1 1 1 0 0 1 1 1
0 1 1 0 0 0 0 1 1 1 1 0 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 1 1 1 1
0 0 1 1 0 1 1 1 0 1 0 0 1 0 1
0 0 1 1 0 1 1 1 0 1 0 0 1 0 1
0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
1 1 0 0 0 1 1 0 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 1 1 1 1 1 0 1 1 1 1 0

total steps = 29

Step1 : (1, 1, SE)
Step2 : (2, 2, NE)
Step3 : (1, 3, E)
Step4 : (1, 4, E)
Step5 : (1, 5, SW)
Step6 : (2, 4, SE)
Step7 : (3, 5, W)
Step8 : (3, 4, SW)
Step9 : (4, 3, S)
Step10 : (5, 3, SW)
Step11 : (6, 2, S)
Step12 : (7, 2, SW)
Step13 : (8, 1, SE)
Step14 : (9, 2, SE)
Step15 : (10, 3, E)
Step16 : (10, 4, NE)
Step17 : (9, 5, NE)
Step18 : (8, 6, E)
Step19 : (8, 7, SE)
Step20 : (9, 8, S)
Step21 : (10, 8, SE)
Step22 : (11, 9, E)
Step23 : (11, 10, NE)
Step24 : (10, 11, E)
Step25 : (10, 12, E)
Step26 : (10, 13, NE)
Step27 : (9, 14, SE)
Step28 : (10, 15, S)
Step29 : (11, 15, S)
Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.
```

(b)
Step1 : (1, 1, SE)
Step2 : (2, 2, E)
Step3 : (2, 3, E)
Step4 : (2, 4 ,S)
Step5 : (3, 4, SW)
Step6 : (4, 3, S)
Step7 : (5, 3, SW)
Step8 : (6, 2, SW)
Step9 : (7, 1, S)

Step10 : (8, 1, SE)
Step11 : (9, 2, SE)
Step12 : (10, 3, E)
Step13 : (10, 4, NE)
Step14 : (9, 5, NE)
Step15 : (8, 6, E)
Step16 : (8, 7, SE)
Step17 : (9, 8, S)
Step18 : (10, 8, SE)
Step19 : (11, 9, E)
Step20 : (11, 10, NE)
Step21 : (10, 11, E)
Step22 : (10, 12, E)

Step23 : (10, 13, E)
Step24 : (10, 14, E)

Step25 : (10, 15, S)
Step26 : (11, 15, S)

(c)
Function Path() finds a path in the maze by the following method:
Starting from position (1, 1), pick the direction “N”, north, as the direction we want the probe to move along and push this step into the stack.
If heading “N” is feasible, then the probe just keeps moving at the direction “N” and push feasible steps into the stack.
If the probe runs into the wall when moving in the direction “N”, Path() will ask the

probe to move in the direction 45 degrees clockwise of “N”, which is “NE”, in the next attempt. During each attempt, whenever the probe runs into a wall, Path() will change the direction of the probe clockwise and try to find a feasible path.
If all the directions are blocked, the stack will pop out all the steps until the one that still has other directions to try.

Since Path() tries out all the directions clockwise starting from “N”, some of the steps where the probe can just directly move in a straight line will be replaced by moving two diagonal lines, which might create redundant steps. For instance, simply moving “E” will be replaced by moving “NE” then “SE”, which is the difference between the path trace by me and the one by the function Path().