EECS2040 Data Structure Hw #2 (Chapter 3 Stack/Queue)
due date 4/17/2021
by 107061123, 孫元駿

Part 1 (50%)

1. (30%) A linear list is being maintained circularly in an array with front and rear set up as for circular queues.
   (a) Obtain a formula in terms of the array capacity, front, and rear, for the number of elements in the list.
   Ans:
   There are three possible situations.
   (1) rear < front
   If rear < front, then the formula is
   $$num\ of\ elements = capacity + front - rear$$
   (2) rear > front
   If front > rear, then the formula is
   $$num\ of\ elements = rear - front$$
   (3) rear == front
   If rear == front, then we can't sure the array is full or empty, so the formula is
   $$num\ of\ elements = 0\ or\ capacity$$

   (b) Assume the kth element in the list is to be deleted, the elements after it should be moved up one position. Give a formula describing the positions of those elements to be moved up one position, i.e., from ??? to ???.
   Ans:
   If ((front + k) % capacity) == rear,
   then don't need to move anything.
   If ((front + k) % capacity) < rear,
   from ((front + k + 1) % capacity) to rear.
   If ((front + k) % capacity) > rear,
   from ((front + k + 1) % capacity) to capacity – 1 and from 0 to rear.

   (c) Assume that we want to insert an element y immediately after the kth element and array doubling is needed (as **Program 3.11** shows or pptx page 83 **void Queue<T>::**Push(**const T&** x) shows), please explain the code using a graphical illustration and explanation.
   Ans:

If $(rear+1) \% capacity == front$

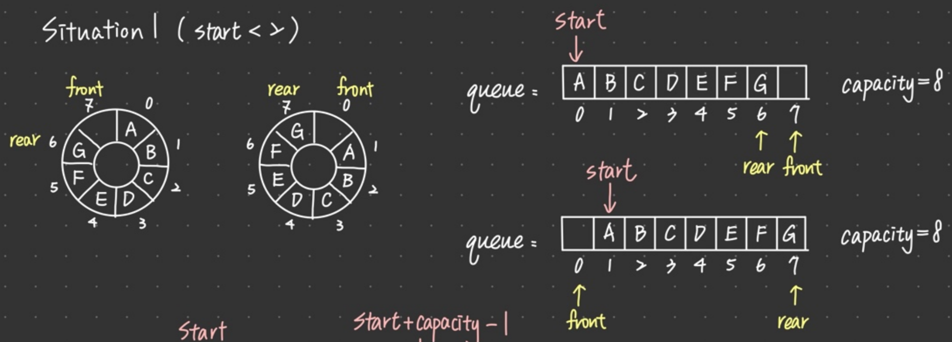
because we can't know if the Queue is full or empty when rear == front

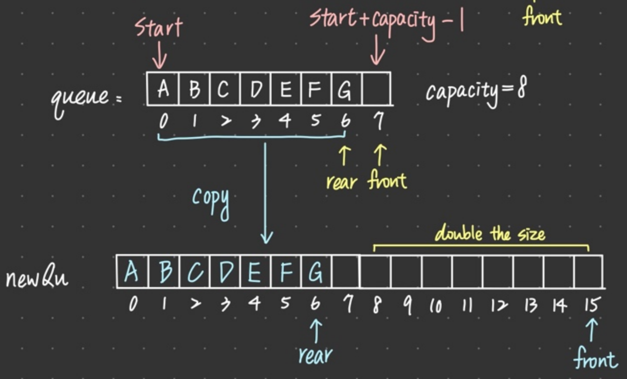
so if we want to insert more element in Queue

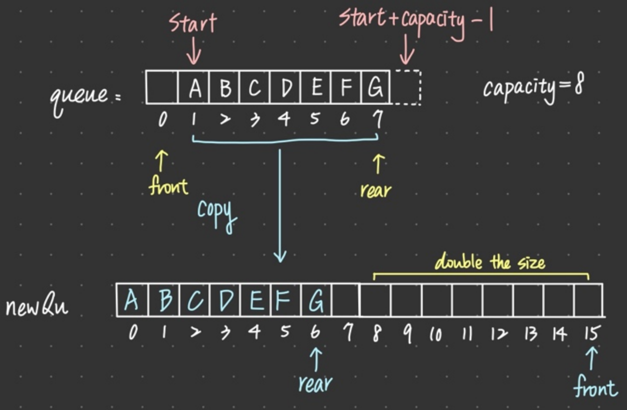
we need to resize the array (Queue), usually we double the size.

Let $start = (front+1) \% capacity$

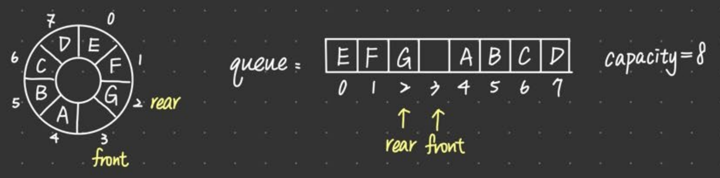
Situation 1 (start < 2)

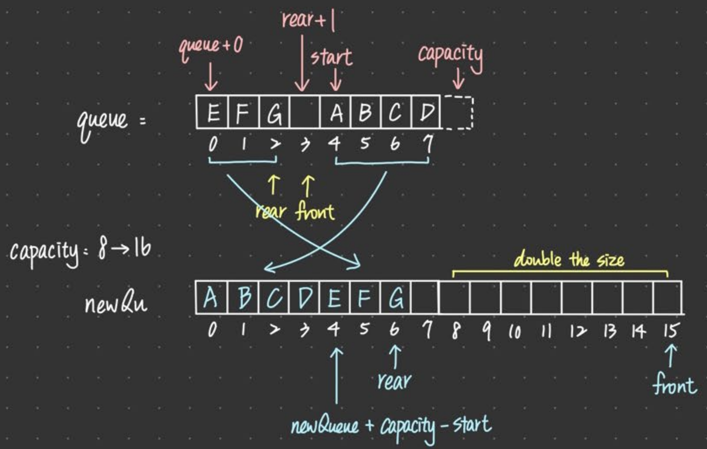




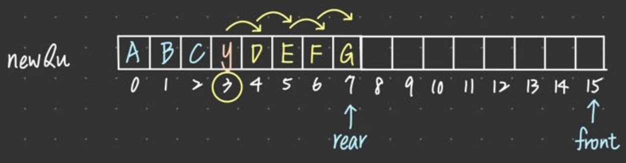


Situation 2 (start ≥ 2)





Then two situation will become the same newQu
so that we can discuss insert part together.
Insert an element y immediately after the kth element * Suppose k = 3



move k to rear, move back one position, and rear = rear+1
then insert y after k element.

2. (20%) Using the operator priorities of Figure 3.15 (or pptx page 130 **Parentheses Handling)** together with those for ‘(‘ and ‘#’ to answer the following:
   (a) In function Postfix (Program 3.19, pptx **Infix to Postfix Algorithm**), what is the maximum number of elements that can be on the stack at any time if the input expression has n operators and delimiters?
   Ans: The maximum number of elements is n + 1 (include ‘#’ and n operators and delimiters)
   (b) What is the answer to (a) if the input expression e has n operators and the depth of nesting of parentheses is at most 6?
   Ans: The maximum number of elements is n+7 (include ‘#’ and ‘(’ * 6 and operator * n)

3. (50%) Write the postfix form and prefix form of the following infix expressions:
   (a) –A + B – C + D
   Ans:
   Postfix: 0 A – B + C – D +
   Prefix: + - + - 0 A B C D
   (b) A * -B + C
   Ans:
   Postfix: A 0 B - * C+
   Prefix: + * A – 0 B C
   (c) (A + B) * D + E / (F + A * D) + C
   Ans:
   Postfix: A B + D * E F A D * + / + C +
   Prefix: + + * + A B D / E + F * A D C
   (d) A && B || C || !(E > F)
   Ans:
   Postfix: A B && C || E F > ! ||
   Prefix: || || && A B C ! > E F
   (e) !(A && !((B < C) || (C > D))) || (C < E)
   Ans:
   Postfix: A B C < C D > || ! && ! C E < ||
   Prefix: || ! && A ! || < B C > C D < C E

Part 2 Coding (50%)

You should submit:

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

(b) Show the execution trace of your program.

1. (30%) Based on the circular queue and template queue ADT in **ADT 3.2** in textbook (or pptx pp.79), write a C++ program to implement the queue ADT**.** Then add two more functions to
   (a) Return the size and capacity of a queue.
   (b) Merge two queues into a one by alternately taking elements from each queue. Te 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.

2. (35%) Referring to **Program 3.13** in textbook (pptx pp.94),
   (a) 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) 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.
   (c) 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 $\Theta(1)$.
   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.

3. (35%) 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.

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

(b) Find a path through the maze shown **Figure 3.14** in textbook.

(c) Trace out the action of function path (**Program 3.16**) on the maze shown. 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？）

入口	0	0	0	0	0	0	0	0	1	
	1	1	1	1	1	1	1	1	0	
	1	0	0	0	0	0	0	0	1	
	0	1	1	1	1	1	1	1	1	
	1	0	0	0	0	0	0	0	1	
	1	1	1	1	1	1	1	1	0	
	1	0	0	0	0	0	0	0	1	
	0	1	1	1	1	1	1	1	1	
	1	0	0	0	0	0	0	0	0	出口

Figure 3.14：唯一路徑的迷宮圖