OOP - Midterm 2 – 2021.5.20

Rules

Exam Time: 5/27(THU) 18:30 ~ 21:20

1.There are four cases for scoring:

If you upload the file :

(1) Before 05/27 21:20 , your highest score will be 100.
(2) 05/27 21:20 ~ 24:00 , your score will reduce 40 points.(highest score will be 60)
(3) 05/28 00:00 ~ 21:20 , your score will reduce 60 points.(highest score will be 40)
(4) After 05/28 21:20, your score will be 0.

2.During exam time, you must join Google Meet room and turn on your camera.
We will announce the room URL before the exam start.

3.You can upload your answer and leave after 21:10, before that, you must stay in the Google Meet room

4.You can download your lab codes or assignment codes from new E3 as reference.

5.You can search online resources as reference. Do not copy code form the internet directly or your score will be 0.

6.If you copy other student's code or let other students copy your code, your score will be 0.

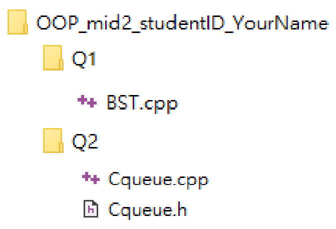
7.We will create a Google doc for you to ask question. Do not use mic to ask.

8. Please use VS2010/VS2017/VS2019 to write your code, if you can’t use them, specify what environment did you use. Create a text file and write down your environment, upload your answer together with your text file.
If your source code cannot be built and you didn’t tell us your building environment, your score will be 0.

Submission detail

1. Turn in:

- Make your directory as below



```
OOP_mid2_studentID_YourName
    Q1
        BST.cpp
    Q2
        Cqueue.cpp
        Cqueue.h
```

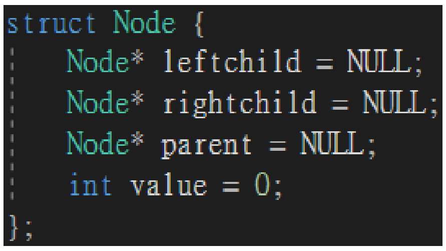
- There should be your answer file (.cpp) in each directory.
- Zip all files in one file and name it OOP_mid2_studentID_YourName.zip

2. You must add a text file to specify what environment do you use if you didn’t use VS2010/VS2017/VS2019.

Q1. Binary Search Tree & Tree Traversal (50 points)

In question 1, you have to implement a Binary Search Tree (BST). Each node has a value. The node in the BST must be greater than or equal to any node in its left sub-tree, and be smaller than any node in its right sub-tree. **All you have to do is finish all TODO parts in BST.cpp and DO NOT modify main.cpp as well as BST.h.** The functions are listed below.

Node is defined:

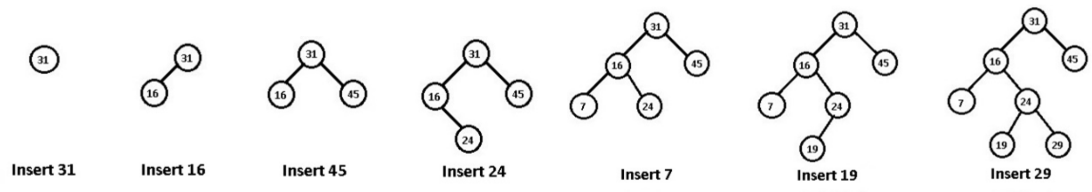


```
struct Node {
    Node* leftchild = NULL;
    Node* rightchild = NULL;
    Node* parent = NULL;
    int value = 0;
};
```

- ➢ void InsertNode(int *value*): Insert a node with *value* to the BST.
- ➢ void DeleteNode(int *value*): Delete a node with *value* in the BST.
- ➢ int ComputeHeight(Node* *node*): Compute the height of the BST.
- ➢ void InOrderPrint(Node* *node*): Use In-Order method to print the BST.
- ➢ void LevelOrderPrint(): Use Level-Order method to print the BST.
- ➢ void Destroy(Node* *node*): Delete all the nodes in the BST rooted at *node*.
- ➢ Node* Search(int *value*): Search the node with *value*.
- ➢ Node* Successor(Node* *node*): Find the node with minimum value in the right sub-tree of *node*.

● Functions explanation

➢ InsertNode(int *value*):
You should create a node with *value*, find the correct position to insert, and set the proper relationship between two nodes. If the BST is empty, the first node is the root node.



➢ DeleteNode(int *value*):
First, you need to search the node with *value*. If the node doesn't exist, you must print "{ *value* } is not found." (E.g.: 2 is not found.). Otherwise, delete the node and adjust the node relationship to satisfy BST condition. **We give you the detail about how to delete the node in Hint part.**

➢ ComputeHeight(Node* *node*):
Compute the height of the BST with the given node. The height of root node is 1.

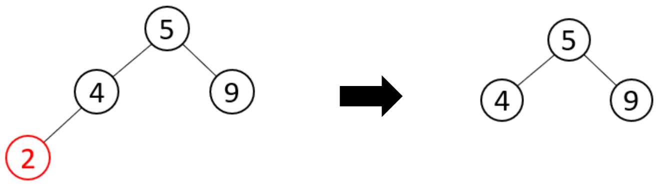
➢ InOrderPrint(Node* *node*):
Print the BST with in-order method from the given node. If you implement this

function correctly, the result will be the number listed in ascending order.

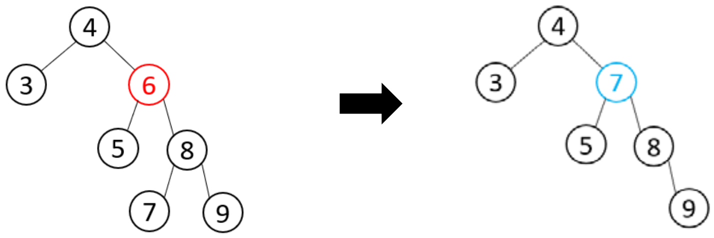
- LevelOrderPrint():
  Print the BST with level-order method.
- Destroy(Node* *node*):
  Delete all the nodes in the BST. You must delete the node **in post-order**. You have to print the value of the node before delete it. This function is called by ~BST().
- Search(int *value*):
  Search the node with *value* in the BST. If the node doesn't exist, return NULL. Otherwise, return the node.
- Successor(Node* *node*):
  Find the node with minimum *value* in the right sub-tree of the given node.

Hint

1. How to delete the node:
   DEL_NODE is the node we want to delete.
   Case 1: DEL_NODE has no left child and right child.
   E.g.: delete 2

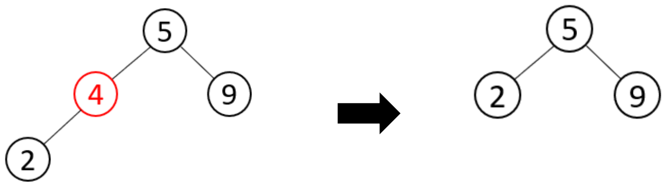


   Case 2: DEL_NODE has left child and right child.
   E.g.: delete 6



   After delete node 6, you have to find a successor to replace the DEL_NODE.
   In this case, the right sub-tree of DEL_NODE 6 are node 7, 8 and 9. The successor is node 7 since it has the minimum value in the right sub-tree.
   You must implement the function Successor(Node* *node*) to find the successor.

   Case 3: DEL_NODE only has left child or right child.
   E.g.: delete 4



   You have to carefully consider all possible conditions in case 2 and case 3.
2. When you implement in-order, level-order and post-order, you can use in-built data structure in STL such as queue, vector.

- ## Input explanation

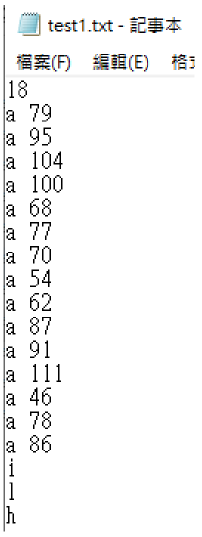
You need to read a text file to get the commands (Table 1).
The first line is the number of commands.
The rest are commands.
Command types:

Command	**Operation**
'a' { value }	InsertNode(value)
'd' { value }	DeleteNode(value)
'i'	InOrderPrint(rootNode)
'l'	LevelOrderPrint()
'h'	ComputeHeight(rootNode)

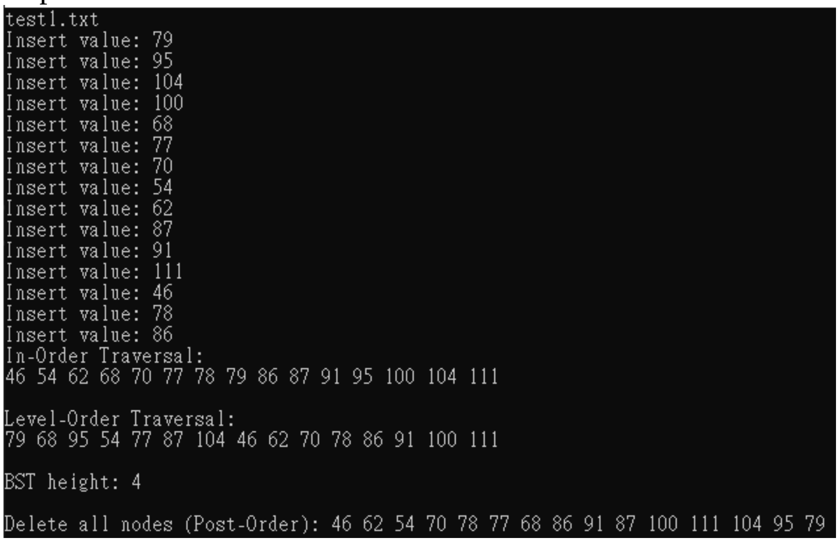
Table 1

- ## Sample Input / Output

1. test1, insert value and print BST in different order

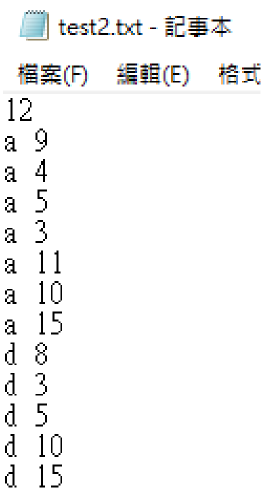


```
18
a 79
a 95
a 104
a 100
a 68
a 77
a 70
a 54
a 62
a 87
a 91
a 111
a 46
a 78
a 86
i
l
h
```

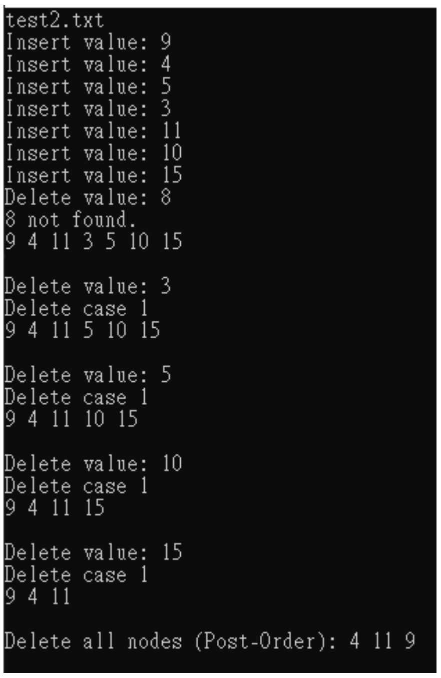


```
test1.txt
Insert value: 79
Insert value: 95
Insert value: 104
Insert value: 100
Insert value: 68
Insert value: 77
Insert value: 70
Insert value: 54
Insert value: 62
Insert value: 87
Insert value: 91
Insert value: 111
Insert value: 46
Insert value: 78
Insert value: 86
In-Order Traversal:
46 54 62 68 70 77 78 79 86 87 91 95 100 104 111

Level-Order Traversal:
79 68 95 54 77 87 104 46 62 70 78 86 91 100 111

BST height: 4

Delete all nodes (Post-Order): 46 62 54 70 78 77 68 86 91 87 100 111 104 95 79
```

2. Delete node: case 1

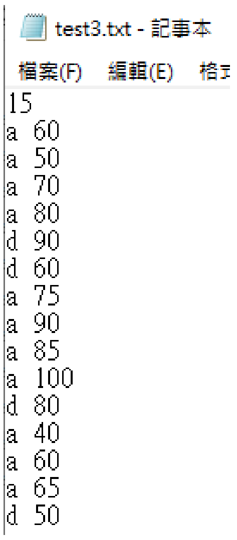


```
12
a 9
a 4
a 5
a 3
a 11
a 10
a 15
d 8
d 3
d 5
d 10
d 15
```

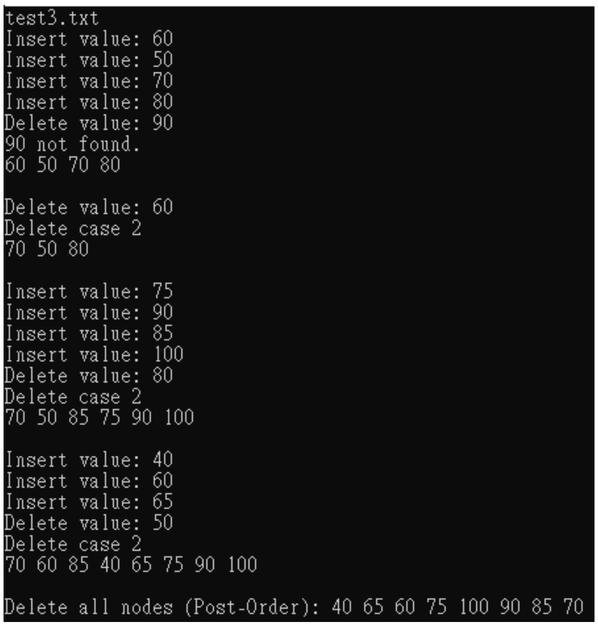


```
test2.txt
Insert value: 9
Insert value: 4
Insert value: 5
Insert value: 3
Insert value: 11
Insert value: 10
Insert value: 15
Delete value: 8
8 not found.
9 4 11 3 5 10 15

Delete value: 3
Delete case 1
9 4 11 5 10 15

Delete value: 5
Delete case 1
9 4 11 10 15

Delete value: 10
Delete case 1
9 4 11 15

Delete value: 15
Delete case 1
9 4 11

Delete all nodes (Post-Order): 4 11 9
```

3. Delete node: case 2

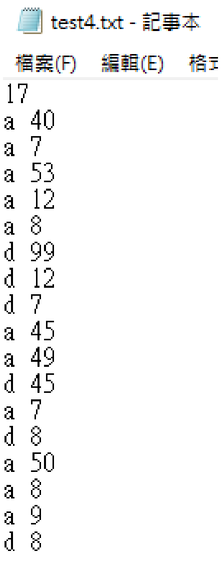


```
15
a 60
a 50
a 70
a 80
d 90
d 60
a 75
a 90
a 85
a 100
d 80
a 40
a 60
a 65
d 50
```

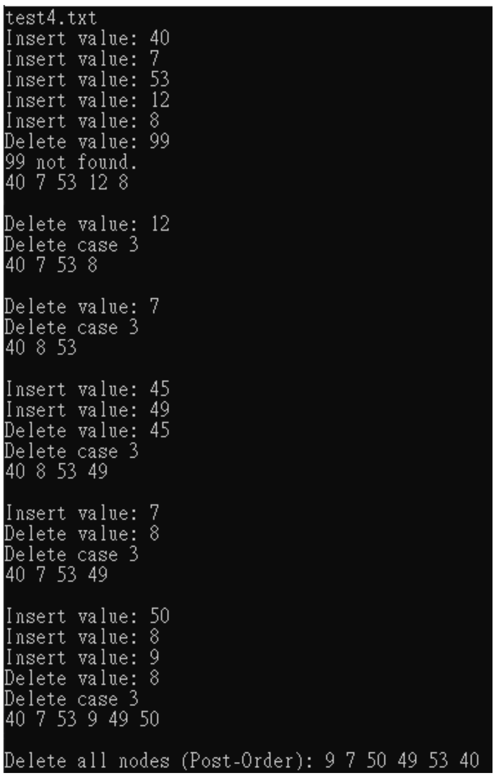


```
test3.txt
Insert value: 60
Insert value: 50
Insert value: 70
Insert value: 80
Delete value: 90
90 not found.
60 50 70 80

Delete value: 60
Delete case 2
70 50 80

Insert value: 75
Insert value: 90
Insert value: 85
Insert value: 100
Delete value: 80
Delete case 2
70 50 85 75 90 100

Insert value: 40
Insert value: 60
Insert value: 65
Delete value: 50
Delete case 2
70 60 85 40 65 75 90 100

Delete all nodes (Post-Order): 40 65 60 75 100 90 85 70
```

4. Delete node: case 3



```
17
a 40
a 7
a 53
a 12
a 8
d 99
d 12
d 7
a 45
a 49
d 45
a 7
d 8
a 50
a 8
a 9
d 8
```



```
test4.txt
Insert value: 40
Insert value: 7
Insert value: 53
Insert value: 12
Insert value: 8
Delete value: 99
99 not found.
40 7 53 12 8

Delete value: 12
Delete case 3
40 7 53 8

Delete value: 7
Delete case 3
40 8 53

Insert value: 45
Insert value: 49
Delete value: 45
Delete case 3
40 8 53 49

Insert value: 7
Delete value: 8
Delete case 3
40 7 53 49

Insert value: 50
Insert value: 8
Insert value: 9
Delete value: 8
Delete case 3
40 7 53 9 49 50

Delete all nodes (Post-Order): 9 7 50 49 53 40
```

● Note

1. Please implement the BST **BY YOURSELF**.
2. If you want to check certain test case, you can change the file_number and index in the main.cpp. **Don't change the rest of code in main.cpp.**
   E.g.: if you only want to check the test case 2, set file_number = 1 and index = 2
3. We will use our main.cpp and BST.h to demo your code. Therefore, **it is important that you should not modify main.cpp (except file_number & index) and BST.h.**
   E.g.: Change function name or function parameters in BST.h is not allowed.

4. All node inserted in the BST will have a unique value.
5. The BST will have at least one node (root) after deleting the node. You don't have to consider the empty tree case.

● Score Distribution (50 points)

If your output format is not same as (similar to) we asked, you will not get any score. Observe sample output carefully and double check your output result.
If your program crash when we test it with our test case, you will not get any points.
We will prepare five test cases to demo your code and compare your output results to our answers.

1. The first two test cases (30 points):

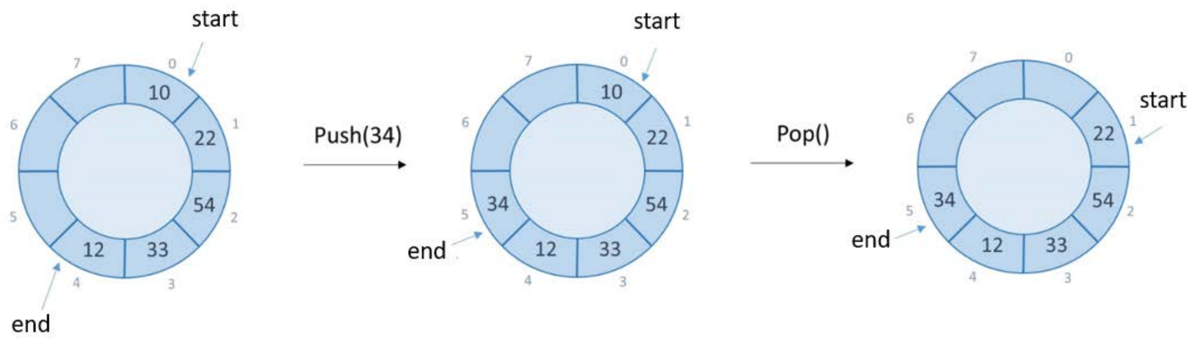
There are no delete cases in the first two test cases.
Each test case is 15 points.
If you can't insert the node to the BST correctly, you will not get any points in this part.
Print height of the BST: 3 points
Print BST in-order: 4 points
Print BST level-order: 4 points
Print BST post-order & Destroy: 4 points

2. The Last three test cases (20 points):

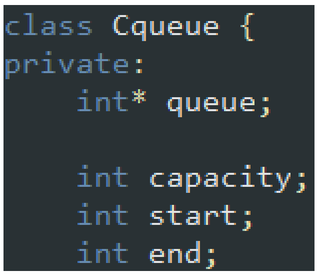
There are three delete cases in the last three test cases respectively.
If you can't insert the node to the BST and print BST in level-order correctly, you will not get any points in this part.
Search node: 3 points
Delete case 1: 3 points
Delete case 2: 7 points
Delete case 3: 7 points

Q2. Circular Queue (50%)

Circular Queue (queue) is a linear data structure in which the operations are performed based on the FIFO (First In First Out) principle and the last element is connected back to the first element to make a circle.



In this question, you are asked to use **Dynamic Allocation of Arrays** to implement a circular queue.



```
class Cqueue {
private:
    int* queue;

    int capacity;
    int start;
    int end;
```

queue: the array that contains all the elements
capacity: capacity of the queue. For example, if **capacity** is 10, it means that this queue can store at most 10 elements.
start: the first element's index
end: the last element's index
When the queue is empty, the value of **start** MUST be 0, and the value of **end** MUST be -1.

The capacity is the total number of cells in the queue.
The size is the number of cells that have data in them.

- Functions explanation
  - Cqueue q(int *Capacity*): Create a queue of capacity "*Capacity*". That is, the number of cells in this queue is "Capacity".
  - void Enqueue (int *x*): Add an element *x* behind the last element in the queue.
  - void Dequeue(): Remove the first element of the queue. If the queue is empty after this operation, set **start** to 0 and **end** to -1.
  - bool IsEmpty(): Return true if the queue is empty and otherwise false.
  - bool IsFull(): Return true if the queue is full and otherwise false.
  - void Reverse(): Reverse all the elements.
  - int GetSize(): Output and return the size in the queue.
  - int GetStart(): Output and return the value of "**start**".
  - int GetEnd(): Output and return the value of "**end**".
  - void Print(): Output the elements in the queue from **start** to **end**.

Example:

```
Cqueue q(3);          // The capacity of the queue is 3
q.Enqueue(1);         // content of q: 1
q.Enqueue(2);         // content of q: 1 2
q.GetSize();          // return 2 and print the message "The current size of queue is 2"
q.Enqueue(3);         // content of q: 1 2 3
q.GetStart();         // return 0 and print the message "Start: 0"
q.GetEnd();           // return 2 and print the message "End: 2"
q.Reverse();          // content of q: 3 2 1
q.Enqueue(4);         // print the message "Queue is full, cannot add more element"
q.Print();       // print the message "The current content of queue is: 3 2 1"
```

● Note

1. You cannot use any built-in data structure implemented by the standard library.
2. You must use **Dynamic Allocation of Arrays** to solve this problem, otherwise your score is 0.
3. You cannot use any built-in data structure such as <queue>, <vector>, <stack>,...
4. Do not modify the content in "int main()".

You can add extra variables or functions, but **cannot** add anything in "int main()".

● Score Distribution (Total: 50 points)

Constructor: 5 points
Enqueue: 5 points
Dequeue: 5 points
IsEmpty: 5 points
IsFull: 5 points
Reverse: 5 points
GetSize: 5 points
GetStart: 5 points
GetEnd: 5 points
Print: 5 points

● Input Format

Read the input data from "Q2_input.txt".

The first line shows the number of test cases.
Each case includes three lines.
The first line is an integer which represents the capacity of the circular queue.
The second line is an integer which represents the number of operations.
In the third line, follow the following rules:
There will be a character representing each operation(e, d, p, r, s, a, b).
See Table2

Command	Operation
e { value }	Enqueue(value)
d	Dequeue()
p	Print()
r	Reverse()
s	GetSize()
a	GetStart()
b	GetEnd()

Table 2

● Output Format

operation	condition and output message
e	If the queue is currently full, print “Queue is full, cannot add more elements.” and otherwise output nothing.
d	If the queue is currently empty, print the “Queue is empty.” and otherwise output nothing.
p	Print the elements in the circular queue from **start** to **end**. There is a whitespace character between two numbers.
r	Do not print anything.
s	If the size of the queue is **n**, print “The size of queue is: **n**”.
a	Print the value of “**start**”. For example, if **start**=5, then print “Start: 5”.
b	Print the value of “**end**”. For example, if **end**=5, then print “End: 5”.

If you do not follow the output format, you will get 0 points.
See more details from the sample input and output.

● Sample Input (Q2_input.txt)

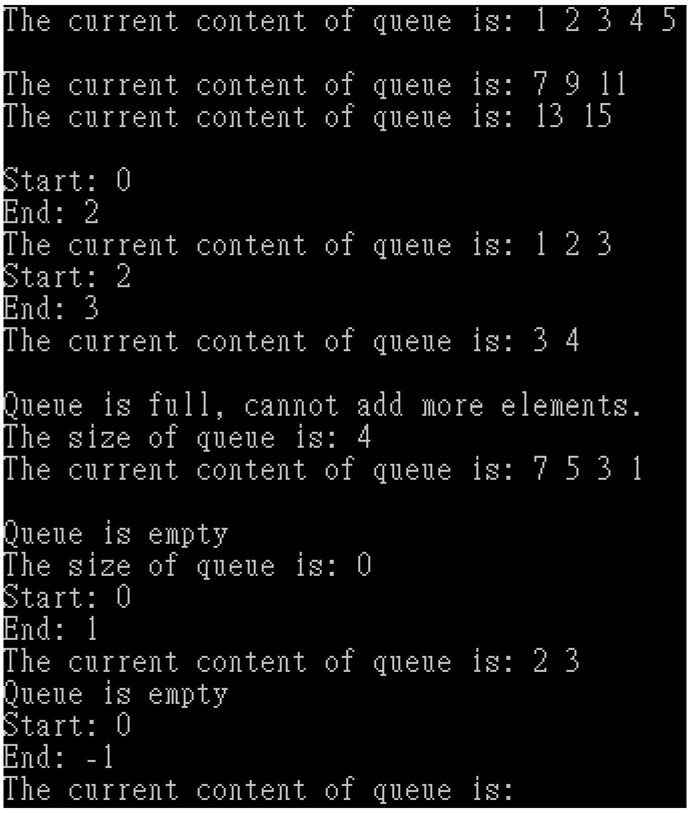
```
5

5
6
e 1 e 2 e 3 e 4 e 5 p

6
10
e 7 e 9 e 11 p d d e 13 e 15 d p
```

4
12
e 1 e 2 e 3 a b p d d e 4 a b p

4
10
e 7 e 1 d e 3 e 5 e 7 e 9 r s p

3
16
e 1 d d s e 2 e 3 a b p d d d r a b p

● Sample Output



```
The current content of queue is: 1 2 3 4 5

The current content of queue is: 7 9 11
The current content of queue is: 13 15

Start: 0
End: 2
The current content of queue is: 1 2 3
Start: 2
End: 3
The current content of queue is: 3 4

Queue is full, cannot add more elements.
The size of queue is: 4
The current content of queue is: 7 5 3 1

Queue is empty
The size of queue is: 0
Start: 0
End: 1
The current content of queue is: 2 3
Queue is empty
Start: 0
End: -1
The current content of queue is:
```