Tutorial 3: Queue, Recursion

Part 1. Queue

1. What would be the value of queues Q1, Q2, and stack S, after the following segment?

```plaintext
1 S = createStack
2 Q1 = createQueue
3 Q2 = createQueue
4 enqueue (Q1, 5)
5 enqueue (Q1, 6)
6 enqueue (Q1, 9)
7 enqueue (Q1, 0)
8 enqueue (Q1, 7)
9 enqueue (Q1, 5)
10 enqueue (Q1, 0)
11 enqueue (Q1, 2)
12 enqueue (Q1, 6)
13 loop (not emptyQueue (Q1))
   1 dequeue (Q1, x)
   2 if (x == 0)
      1 x = 0
      2 loop (not emptyStack (S))
         1 popStack(S, &y)
         2 z = z + y
      3 end loop
   4 enqueue (Q2, z)
   3 else
      1 pushStack (S, x)
      4 end if
14 end loop
```

2. What would be the contents of queue Q after the following code is executed and the following data are entered?

```plaintext
1 Q = createQueue
2 loop (not end of file)
   1 read number
   2 if (number not 0)
      1 enqueue (Q, number)
   3 else
      1 queuerear (Q, x)
      2 enqueue (Q, x)
   4 end if
3 end loop
```

The data are: 5, 7, 12, 4, 0, 4, 6, 8, 67, 34, 23, 5, 0, 44, 33, 22, 6, 0

3. What would be the contents of queue Q1 after the following code is executed and the following data are entered?

```plaintext
1 Q1 = createQueue
2 S1 = createStack
3 loop (not end of file)
   1 read number
```
2 if (number not 0)
1 pushStack (S1, number)
3 else
1 popStack (S1, x)
2 popStack (S1, x)
3 loop (not empty S1)
1 popStack (S1, x)
2 enqueue (Q1, x)
4 end loop
4 end if
4 end loop

The data are: 5, 7, 12, 4, 0, 4, 6, 8, 67, 34, 23, 5, 0, 44, 33, 22, 6, 0

4. Imagine that the contents of queue Q1 and queue Q2 are as shown. What would be the contents of queues Q1, Q2 and Q3 after the following code is executed? The queue contents are shown front (left) to rear (right).

Q1: 42 30 41 31 19 20 25 14 10 11 12 15
Q2: 4 5 4 10 13

1 Q3 = createQueue
2 count = 0
3 loop (not empty Q1 and not empty Q2)
1 count = count + 1
2 dequeue (Q1, x)
3 dequeue (Q2, y)
4 if (y equal count)
1 enqueue (Q3, x)
5 end if
4 end loop

5. Write the pseudocode for a function named reverseStack that reverses the contents of a stack. The algorithm must have only one parameter for the stack to be reversed. Hint: use a temporary queue and the appendix

6. Write the pseudocode for a method of the Linked Queue in the lecture notes (implemented with two pointers front and rear) that receives another linked queue and appends the input queue to the end of the current queue. The input queue will be empty afterward.

algorithm appendQueue (ref in_queue <Linked Queue>)
This algorithm appends the input in_queue to the end of this queue
Pre in_queue is a linked queue
Post the in_queue will be appended into this queue and be empty

end appendQueue
Part 2. Recursion

7. Consider the following algorithm:

What would be returned if fun1 is called as
a. fun1 (4)?
b. fun1 (10)?
c. fun1 (12)?

8. Consider the following algorithm:

What would be returned if fun3 is called as
a. fun3 (10, 4)?
b. fun3 (4, 3)?
c. fun3 (4, 7)?
d. fun3 (0, 0)?

9. Ackerman’s number, used in mathematical logic, can be calculated using the formula shown in the figure. Write a recursive algorithm that calculates Ackerman’s number. Verify your algorithm by using it to manually calculate the following test cases: Ackerman (2, 3), Ackerman (2, 5), Ackerman (0, 3), and Ackerman (3, 0).

10. The following recursive algorithm converts a string of numerals to an integer. For example, “43567” will be converted to 43567.

Note that, the reference parameter mul is updated (in step 3) when this function is called recursively (in step 3) and is used in the caller function (in step 4).

Run the algorithm step-by-step to demonstrate its correctness with the following string:

a) “1234”
b) “2030”
Appendix – Two ways of queue implementation

Basically, the principle of a queue is first in first out. However, regarding practical aspect, there are two ways to get a queue implemented: using contiguous array and linked list.

Queue implementation based on contiguous array (as a circular array)

A contiguous queue can be generally declared as follows:

class Queue
front <int>
rear <int>
array[maxSize] <data>

where front and rear keep positions of the first and the last elements of the queue respectively. Their values are -1 when the queue is empty.

Example 1. Followings are algorithms of insert, remove and getSize methods for the contiguous queue

algorithm insert (val n <data>)
Pre The queue is not full
Post n will be inserted to the queue
1. if(rear == maxSize-1) rear = -1;
2. array[++rear] = n;
3. if (front == -1) front = 0; //the queue was empty before => only element
end insert
data algorithm remove
Pre The queue is not empty
Post Remove the first element of the queue
Return The removed element
1. temp = queArray[front]
2. if (rear == front) {rear = front = -1} //remove the only element => empty
3. else if(front == maxSize-1) front = 0
4. else front++
5. return temp;
end remove

int algorithm getSize
Pre None
Post None
Return The size of queue
1. if(rear >= front) size = rear-front+1;
2. else size = (maxSize-front) + (rear+1)
3. return size;
end getSize

Queue implementation based on linked list

A linked queue can be generally declared as follows:

class Queue
list <Linked_list>
Here, we use a linked list to store the queue elements. Supposed that we have a class `Linked_list` implemented together with class `Node` as follows:

```plaintext
class Linked_list
head <Node*>

class Node
next <Node*>
data
```

**Example 2.** Followings are algorithms of `insert`, `remove` and `getSize` methods for class `Queue`

```plaintext
algorithm insert (val n <data>)
    Pre None
    Post n will be inserted to the queue
1. list.insertLast(n);
end insert
data algorithm remove
    Pre The queue is not empty
    Post Remove the first element of the queue
    Return The removed element
1. return list.removeFirst();
end remove
int algorithm getSize
    Pre None
    Post None
    Return The size of queue
1. return list.getSize();
end getSize
```

Of course, to complete the queue implementation, we must write the additional methods of `insertLast`, `removeFirst` and `getSize` methods for class `Linked_list`. After finishing the previous tutorials and labs, you should be able to complete these methods yourselves.