QueueManager

I created two classes to test QueueManager for performance and memory efficiency.

These are divided into two namespaces:

- DoububleLinkedListQueue::QueueManager
- MemMoveQueue::QueueManager

I. Header

These two methods use a common header area in the buffer.

- From index 0 to 79, 20 queues use 4 bytes each.
    - The first 2 bytes store the front data index of the queue, and the next 2 bytes store the end data index.
- Index 80 contains the index of the rightmost data node in the buffer
- Idex 82 stores the number of data nodes that all queues have.

II. DoububleLinkedListQueue::QueueManger

Like a double linked list, the data used in the queue uses 5 bytes for each node.
The first 1 byte stores char type data, the next 2 bytes store the index of the previous node, and the last 2 bytes store the index of the next node. Therefore, a total of 392 data nodes can be entered into a maximum of 20 queues.

The advantage of this method is that added queue data continues to be stored sequentially to the right of the buffer, regardless of the queue ID. Therefore, the Enqueue function is processed very quickly.

In the case of dequeue, the node on the far right of the buffer is copied to the dequeued node buffer location, and the index values of the previous and next nodes of the copied node are changed, so it is processed relatively quickly.

On the other hand, when DestroyQueue is called, it dequeues as much data as the number of data in the queue, so many node copies and related node updates occur, so frequent deletion of large queues can reduce performance.

III. MemMoveQueue::QueueManager

This method has the advantage that more data can be used in queues by using only 1 byte because each data node uses only char data. Therefore, a total of 1964 data can be pushed into a maximum of 20 queues.

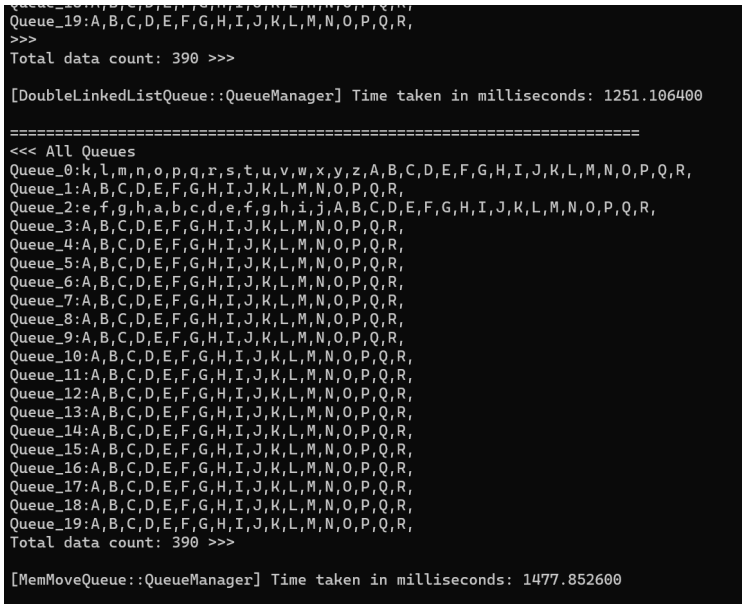
All queue data is gathered in order in the buffer, but not in queue ID order. When data is pushed into the middle of a queue buffer, all buffers behind the used index are moved to the right by one space using the memmove function.

On the other hand, Dequeue works very quickly by clearing only the relevant buffer and changing the start position of the queue, and DestroyQueue also works efficiently because it deletes all collected queue data at once.

IV. Conclusion

I produced and tested it in two ways, and the results came out as expected.
The following is the result of running the submitted Visual Studio project, showing the execution time of the two types of queue managers.

- DoububleLinkedListQueue::QueueManager takes about 1251 ms
- MemMoveQueue::QueueManager takes about 1477 ms



While DoubleLinkedListQueue has fast overall processing speed, it has the disadvantage of reducing the total number of available data to 392.

In contrast, MemMoveQueue has a processing speed of about 20% slower, but has the advantage of being able to use 1964 pieces of total data.

The pros and cons of each are relative to each other, so the choice should be made by considering what functions and input/output patterns are used.