Assignment 2 - Queue class

Necessary skills: *class construction, class composition, pointers and linked lists*, *heap memory management with C++ new/delete operators.*

Description

Create a Queue class that implements a **queue** abstraction. A queue is a FIFO list (First In First Out queue). A simple example is waiting in line, where the first person in the line is the first served. New arrivals are added to the back of the line, the next person served is (removed) from the front of the line.

The Queue class needs to implement the following operations:

- **adding** to the queue at one end (the tail)
- **removing** from the queue at the other end (the head)
- **printing** all items the queue (from head to tail)
- **erasing** all items in the queue (leaving the queue empty).
- destructor to empty the queue before it's destroyed (to release all memory)

Additions and removals always occur at the opposite ends of the queue.

You should create the following methods in your **Queue** class to implement the above operations

- addItem
- removeItem
- print
- erase

Your Queue class must implement a linked list. Linked lists are implemented using individual items which contain a pointer to the next item in the list, as well as the information to be stored.

Your Queue implementation uses a companion **QueueItem** class to represent each element in the list. A QueueItem contains character string as the data value, a unique (among all QueueItems in a Queue) integer item identifier, and a pointer to the next QueueItem in the list. The following is the definition for the QueueItem class.

```
class QueueItem {
public:
    QueueItem(const char *pData, int id); // ctor
    void setNext(QueueItem *pItem);
    QueueItem* getNext() const;
    int getId() const;
    const char* getData() const;

private:
    char _data[30];    // data value (null terminated character string)
    const int _itemId; // unique id for item in queue
    QueueItem* _pNext; // next item in queue
};
```

The QueueItem member functions are very basic, just setting or getting data members of the class. All the linked list manipulation is done by the Queue class member functions.

The Queue class member functions manipulate the linked list of QueueItem's, creating and destroying QueueItem objects as needed using the C++ new and delete operators. The Queue class member data includes a pointer to the head and and pointer to the tail of the linked list of QueueItems, and an integer item counter used to provide a unique item ID for every newly created QueueItem (incremented each time a new QueueItem is added, and passed as a parameter to the QueueItem constructor. It is never decremented).

The following is a partial example of the Queue class; you will need to fill in the remaining methods.

```
class Queue {
public:
    Queue();    // ctor inits a new empty Queue
    ~Queue();   // dtor erases any remaining QueueItems
    void addItem(const char *pData);
    void removeItem();
    ...
private:
    QueueItem *_pHead; // always points to first QueueItem in the list
    QueueItem *_pTail; // always points to the last QueueItem in the list
    int _itemCounter;  // always increasing for a unique id to assign to each new QueueItem
};
```

The Queue class member functions should not have access to the private members of QueueItem objects. They call the public member functions of QueueItem.

As an example, the outline of the `Queue::addItem()` member function is shown below. It must add a new QueueItem at the tail of the Queue, and update the _pTail pointer to point to it. The first item placed in the Queue becomes both the head and the tail of the list.:

```
void Queue::addItem(const char *pData)
{
  // dynamically create and init a new QueueItem object
  QueueItem *pItem = new QueueItem(pData, ++_itemCounter);

  if (0 == _pHead)  // check for empty queue
      _pHead = _pTail = pItem;
  else
  {
     // link new item onto tail of list using _pTail pointer
     ...
  }
}
```

The removeItem() method removes the head QueueItem from the queue, and should release the memory using the C++ delete operator. It updates _pHead to point at the following item (if any) as the new head. If the list becomes empty, both _pHead and _pTail must be set to null (0). It does not change the value of _itemCounter (which is always incremented when a new item is added). If called on an empty Queue, it does nothing.

The erase() method removes all the items in the queue and should release the memory. To implement, you could loop calling removeItem() until the queue is empty.

The Queue destructor should ensure that all items are removed from the queue. The easiest way is to call the erase() method from the destructor.

The user code (main) never see's QueueItem objects, since they are used only for implementation inside of class Queue. main() has only the Queue object to work with. For example, the following code would create a queue with several elements, and then print it out:

```
// note - you may need to change the definition of the main function to
// be consistent with what your C++ compiler expects.
int main() {
  Queue myQueue;

  myQueue.removeItem();
  myQueue.addItem("red");
  myQueue.addItem("green");
  myQueue.addItem("blue");
  myQueue.addItem("orange");
  myQueue.print();  // print contents of queue (item ID and data)

  myQueue.removeItem();
  ...
}
```

Use separate header (.h) and source (.cpp) files for this homework. The .cpp file for a class should #include the corresponding .h file for the class (e.g. QueueItem.cpp should #include "QueueItem.h"). main.cpp should #include "Queue.h".

- QueueItem.h contains the class definition for QueueItem
- QueueItem.cpp contains the member function implementations for QueueItem.
- Queue.h contains the class definition for Queue.
- Queue.cpp contains the member function implementations for Queue.
- main.cpp contains the main() test function.

To test your Queue and QueueItem class implementations, write a main() program which does the following:

- Create a Queue
- Call Remove. Should do nothing since the Queue is empty.
- Add 4 elements.
- Print out the list, both the number and data.
- Remove 2 elements.
- Add 4 elements.
- Print out the list, both the number and data.
- Remove 4 elements
- Print out the list, both the number and data.
- Erase the queue.
- Add 3 elements.
- Print out the list, both the number and data.
- Erase the queue.
- Print out the list.

Print a heading when printing the queue so each of the above print calls is obvious. Use unique data values for each of the items added to the queue.

Additional Information

See the Review material in Week 1 for additional help on pointers, dynamic memory management, and linked lists. Also this document on **Linked List Basics**.

Links

- ReelLearning on YouTube - Data Structures: Introduction to Linked Lists