Paul Nickerson
COP 3530
Section 1087
9/11/2014
Exercise 01

I hereby affirm that the following work is my own and that the Honor Code was neither bent nor broken:

_____

Exercise Summary

I attempted to complete the assignment sequentially, by first drawing the expected buffer state and head/tail index values after various operations, then writing pseudocode (this was the easiest part of the assignment), then writing c++ source code by hand on notebook paper to simulate an exam environment, and finally copying that to the text editor and compiling. The program compiled successfully, but needed a couple of tweaks, particularly to the size calculation method, which did not perform correctly (this was the most difficult part of the assignment). While tweaking the source code, I made notes of what I was doing and why I was doing it in order to internalize the lessons that those corrections taught. In order to improve the project, it would be useful to practice using either the memory library's allocate functionality or the new c++11 shared_ptr, as well as to practice using references instead of pointers.

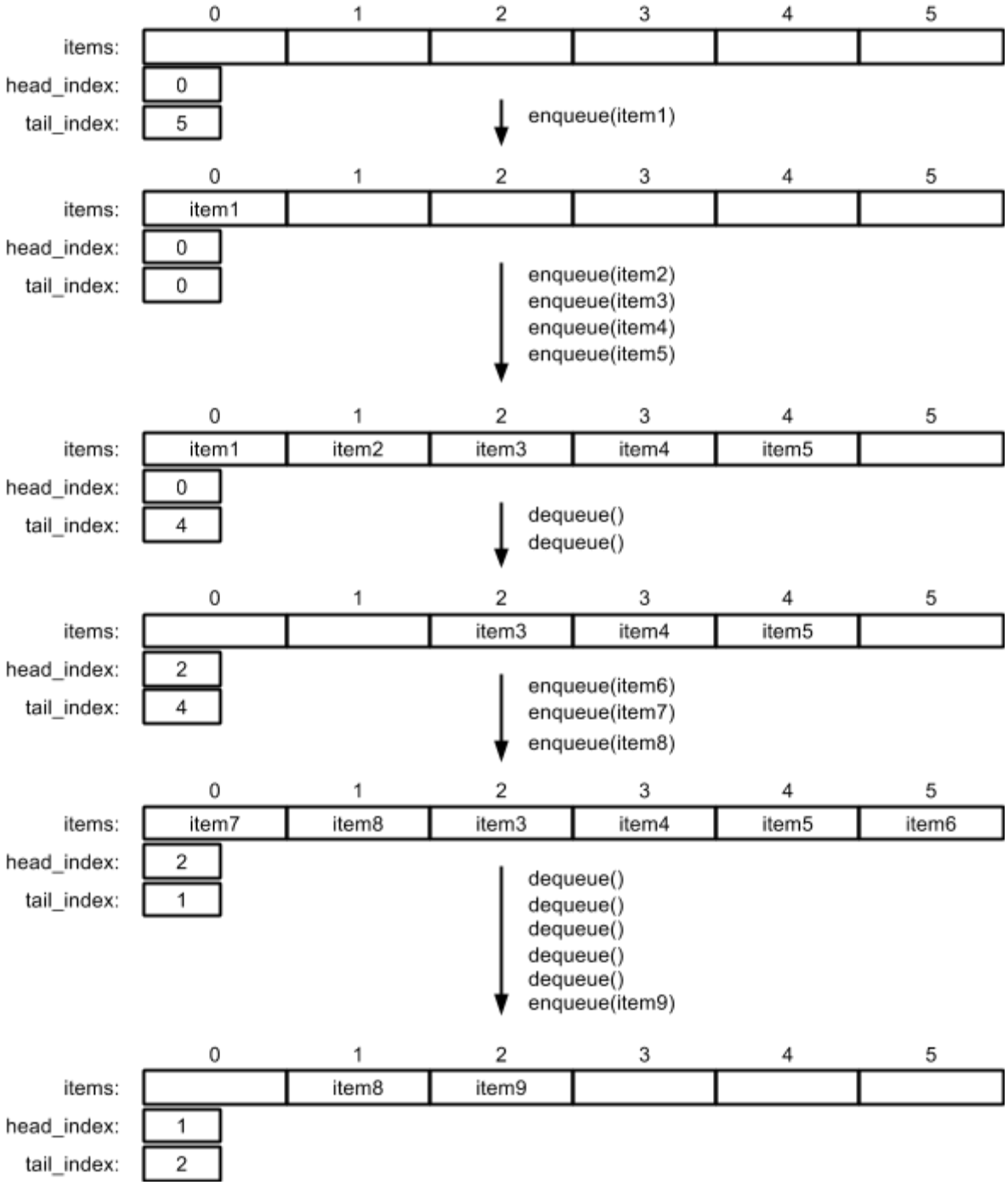
Paul Nickerson
COP 3530
Section 1087
9/11/2014
Exercise 01

Program Questions:

1. Does the program compile without errors?
    a. Yes, the program successfully compiles
2. Does the program compile without warnings?
    a. Yes, there were no warnings given during compilation
3. Does the program run without crashing?
    a. Yes, the program runs and finishes as expected
4. Describe how you tested the program.
    a. I implemented a series of for loops to enqueue and dequeue in an effort to ensure that the tail and head indices make full rotations through the circular buffer, using the assert preprocessor macro to test that dequeueing item values were as expected and that the calculated current size was as expected. I then induced the queue to throw exceptions indicating that the queue was full or empty when attempting to enqueue and dequeue, respectively, which tested the is_full and is_empty methods. Again, within the exception handlers for two operations, I again used assert to determine that the calculated queue size was either the maximum queue size for a full state and zero for an empty state.
5. Describe the ways in which the program does *not* meet assignment's specifications.
    a. The program meets all assignment specifications
6. Describe all known and suspected bugs.
    a. While the queue frees all items in the array when its destructor is called, the queue expects that the client will free items being dequeued when the client is done using them. This is potentially a messy design that can easily lead to memory corruption/leaks.
7. Does the program run correctly?
    a. Yes, the program runs, produces the expected output, and does not fail any of the assertion tests

Part 1

Design



Pseudocode

```
is_queue_empty:
        if (head_index points to the item after tail_index, following circular buffer)
        and (the array element at head_index is a null pointer):
                the array is empty
        else:
                the array is not empty
```

```
is_queue_full:
        if (head_index points to the item after tail_index, following circular buffer)
        and (the array element at head_index is not a null pointer):
                the array is full
        else:
                the array is not full
```

```
compute_queue_current_size:
        if queue is full:
                size = maximum queue size
        else if queue is empty:
                size = 0
        else:
                delta = tail_index - head_index
                delta = delta + 1
                delta = delta + maximum queue size
                size = delta modulo max queue size
```

```
enqueue(item pointer):
        if queue is full:
                throw an error
        increment tail index, rolling over to beginning if necessary
        set array element pointed to by tail to the item pointer
```

```
dequeue():
        if queue is empty:
                throw an error
        save pointer stored in array at head index to temporary variable
        set the head array element to a null pointer
        increment head index, rolling over to beginning if necessary
        return the pointer previously stored in the temporary variable
```

C++ code:

```
#include <stdexcept>
#include <iostream>
#include <cassert>

template<typename Item>
class Queue {
private:
    using item_ptr = Item*;
    int head_index;
    int tail_index;
    const int max_queue_size;
    item_ptr* item_array;
public:
    Queue(const int size): max_queue_size(size) {
        item_array = new item_ptr[max_queue_size];
        head_index = 0;
        tail_index = max_queue_size - 1;
        for (int i = 0; i != max_queue_size; ++i) {
            item_array[i] = nullptr;
        }
    }
    ~Queue() {
        delete[] item_array;
    }
    const bool is_empty() const {
        int next_index_after_tail = (tail_index + 1) % max_queue_size;
        if (head_index == next_index_after_tail
            && item_array[head_index] == nullptr) {
            return true;
        } else {
            return false;
        }
    }
    const bool is_full() const {
        int next_index_after_tail = (tail_index + 1) % max_queue_size;
        if (head_index == next_index_after_tail
            && item_array[head_index] != nullptr) {
            return true;
        } else {
            return false;
        }
    }
    const int compute_current_queue_size() const {
        int size;
        if (is_empty()) {
```

```
            size = 0;
        } else if (is_full()) {
            size = max_queue_size;
        } else {
            int index_delta = tail_index - head_index + 1;
            size = (index_delta + max_queue_size) % max_queue_size;
        }
        return size;
    }
    void enqueue(item_ptr new_item) {
        if (is_full()) {
            throw std::runtime_error("queue is full");
        }
        tail_index = (tail_index + 1) % max_queue_size;
        item_array[tail_index] = new_item;
    }
    item_ptr dequeue() {
        if (is_empty()) {
            throw std::runtime_error("queue is empty");
        }
        item_ptr tmp_head_ptr = item_array[head_index];
        item_array[head_index] = nullptr;
        head_index = (head_index + 1) % max_queue_size;
        return tmp_head_ptr;
    }
};

int main() {
    std::cout << "running tests" << std::endl;
    int max_queue_size = 10;
    Queue<int> q(max_queue_size);
    std::cout << "enqueue: ";
    for (int i = 0; i < 7; ++i) {
        std::cout << i << ", ";
        q.enqueue(new int(i));
    }
    std::cout << std::endl;
    assert(q.compute_current_queue_size() == 7);
    std::cout << "dequeue: ";
    for (int i = 0; i < 5; ++i) {
        int next_item = *(q.dequeue());
        std::cout << next_item << ", ";
        assert(next_item == i);
    }
    std::cout << std::endl;
    assert(q.compute_current_queue_size() == 2);
    std::cout << "enqueue: ";
    for (int i = 0; i < 7; ++i) {
```

```
        std::cout << i << ", ";
        q.enqueue(new int(i));
    }
    std::cout << std::endl;
    std::cout << "dequeue: ";
    assert(q.compute_current_queue_size() == 9);
    while (q.compute_current_queue_size() > 4) {
        int next_item = *(q.dequeue());
        std::cout << next_item << ", ";
    }
    std::cout << std::endl;
    try {
        std::cout << "enqueue: ";
        while (1) {
            std::cout << 31337 << ", ";
            q.enqueue(new int(31337));
        }
    } catch (std::exception e) {
        assert(q.compute_current_queue_size() == max_queue_size);
        std::cout << std::endl;
    }
    try {
        std::cout << "dequeue: ";
        while (1) {
            int next_item = *(q.dequeue());
            std::cout << next_item << ", ";
        }
    } catch (std::exception e) {
        assert(q.compute_current_queue_size() == 0);
        std::cout << std::endl;
    }
    std::cout << "all tests passed" << std::endl;
}
```

Output:

```
running tests
enqueue: 0, 1, 2, 3, 4, 5, 6,
dequeue: 0, 1, 2, 3, 4,
enqueue: 0, 1, 2, 3, 4, 5, 6,
dequeue: 5, 6, 0, 1, 2,
enqueue: 31337, 31337, 31337, 31337, 31337, 31337, 31337,
dequeue: 3, 4, 5, 6, 31337, 31337, 31337, 31337, 31337, 31337,
all tests passed
```