Fernando Salazar

Assignment 3 ReadMe

Based on what we know about linked lists, stacks, and queues, design a linked queue (a queue using a linked-list to store the data in the structure)

Design, implement, and test a Queue data structure that:

uses a linked-list to store values in the queue

has an enqueue method that will appropriately add a value to the back of the queue as an appropriate element

has a dequeue method that will appropriately remove an element from the front of the queue and return its value

Optionally has a peek method that returns the value at the front of the queue without removing it

Bonus if you also create an array based Queue!

Tests: Be sure to include at least one test for each piece of functionality that should verify that your code is working!

Be sure to commit changes regularly to your git repo

Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).

Remember to submit a link to this project in Moodle to remind us to grade it!

Note: This assignment is partly to get you some practice with basic pointers if you have not used them much, to get you thinking about dynamically sizing containers, and to think about what might be efficient and what might be able to be improved.

//Design information

Started with my information from assignment 2.

2:Based on what we know about linked lists, stacks, and queues, design a queue data structure:

2.1What functions are we likely to need for a queue to function like the one discussed in class?

A que is first in first out.

Add function: A function to add a new item to the end of the line.

remove function: A function that will remove the item at is at the front of the line.

empty furntion: A function that will check if the que is empty.

number function: A function that keeps track of the size of the que so it is know what the last item is before

a new one is added or removed.

2.2What values will we need to know about the structure for our queue to function properly?

Count: integer for que size

front object: a value that has the last object in and its number place?

Last object: A value that has what object is in the front of the line?

probably more but these are the obvious ones.

// what changed

add function renamed to enqueue

remove function renamed to dequeue

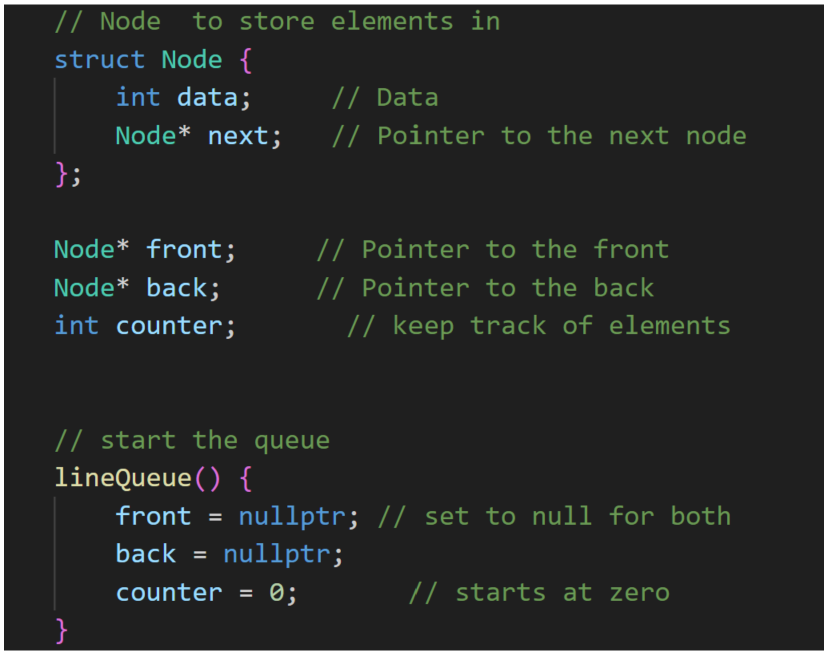
no empty function

added peak function to see values

number function renamed to size   // they should now better follow naming conventions.

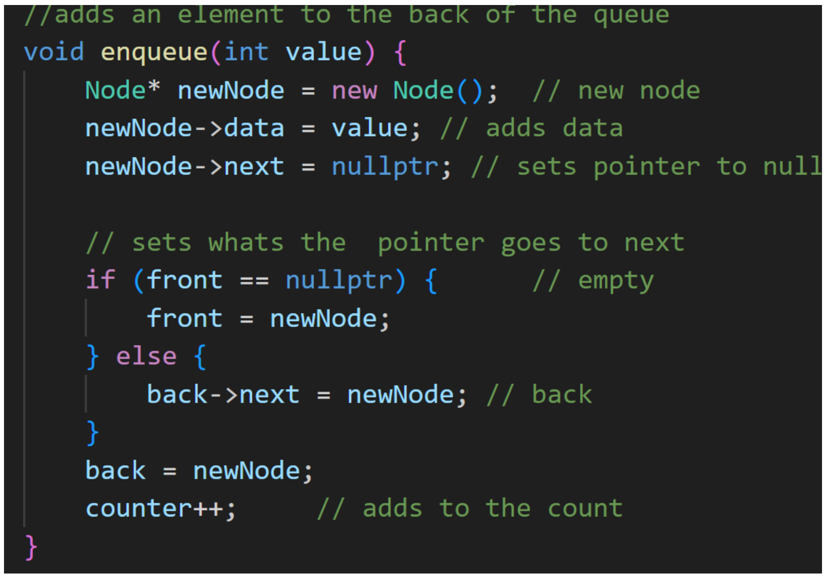
// requirements

linked-list to store values in the queue



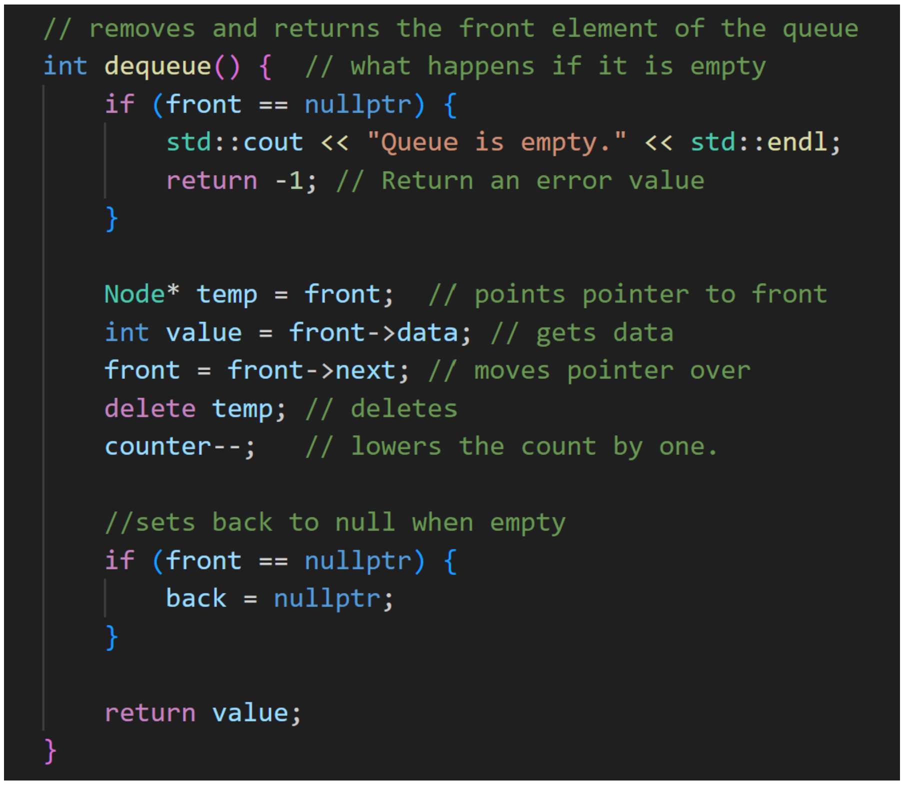
```
// Node  to store elements in
struct Node {
    int data;     // Data
    Node* next;   // Pointer to the next node
};

Node* front;      // Pointer to the front
Node* back;       // Pointer to the back
int counter;        // keep track of elements


// start the queue
lineQueue() {
    front = nullptr; // set to null for both
    back = nullptr;
    counter = 0;        // starts at zero
}
```

// enqueue method



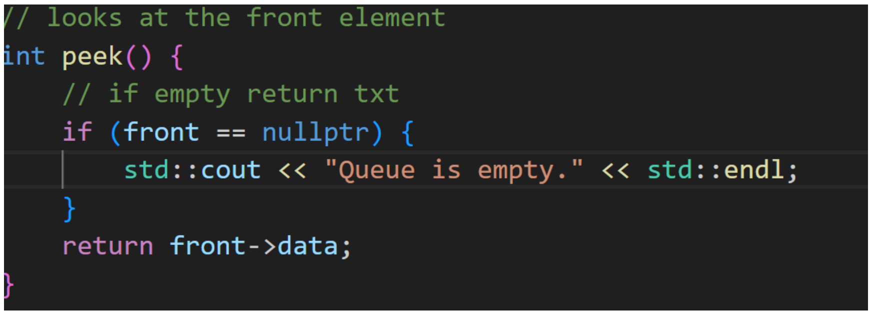
```
//adds an element to the back of the queue
void enqueue(int value) {
    Node* newNode = new Node();  // new node
    newNode->data = value; // adds data
    newNode->next = nullptr; // sets pointer to null

    // sets whats the  pointer goes to next
    if (front == nullptr) {      // empty
        front = newNode;
    } else {
        back->next = newNode; // back
    }
    back = newNode;
    counter++;      // adds to the count
}
```

// dequeue function



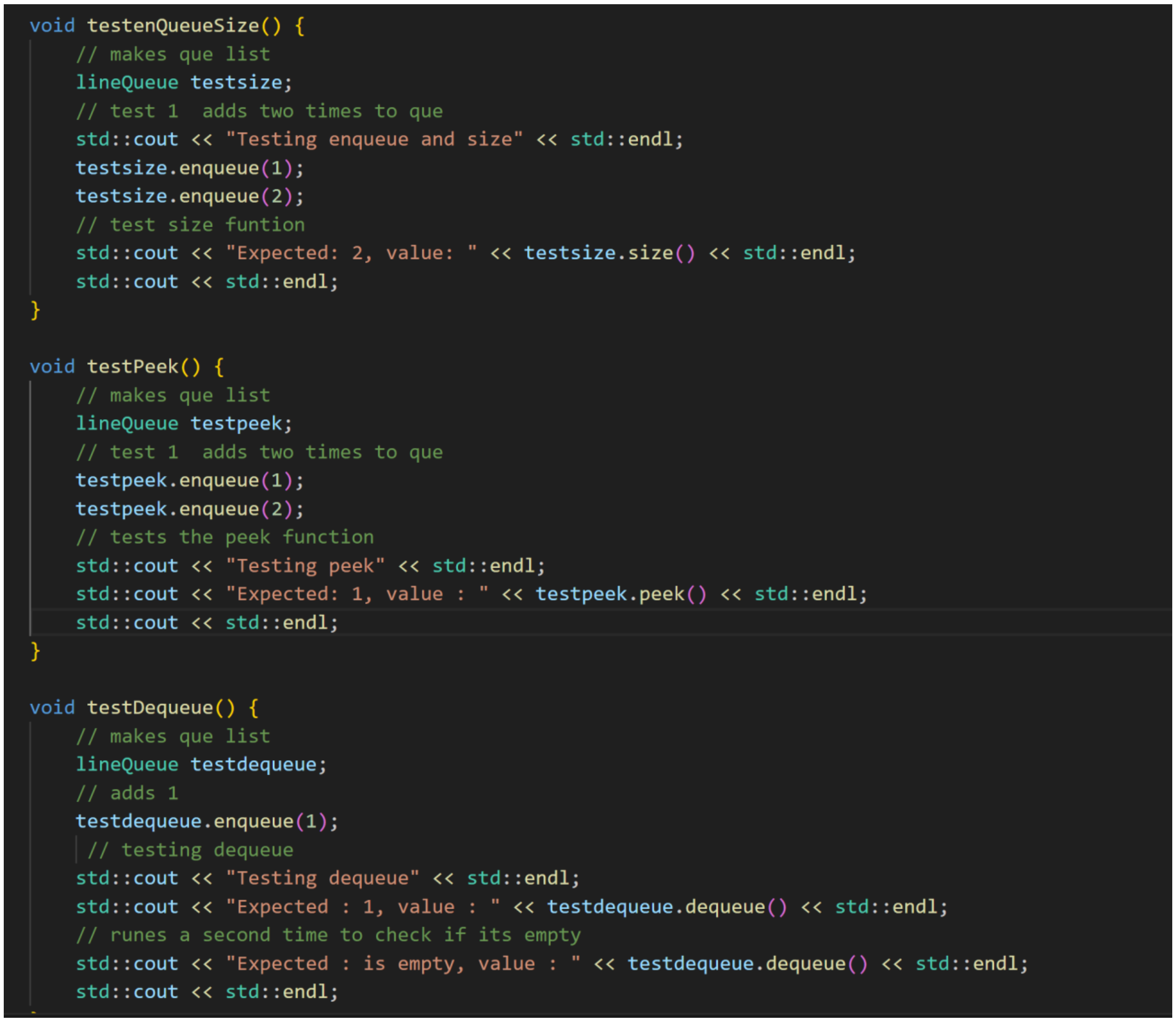
```
// removes and returns the front element of the queue
int dequeue() {  // what happens if it is empty
    if (front == nullptr) {
        std::cout << "Queue is empty." << std::endl;
        return -1; // Return an error value
    }

    Node* temp = front;  // points pointer to front
    int value = front->data; // gets data
    front = front->next; // moves pointer over
    delete temp; // deletes
    counter--;   // lowers the count by one.

    //sets back to null when empty
    if (front == nullptr) {
        back = nullptr;
    }

    return value;
}
```

// peek function



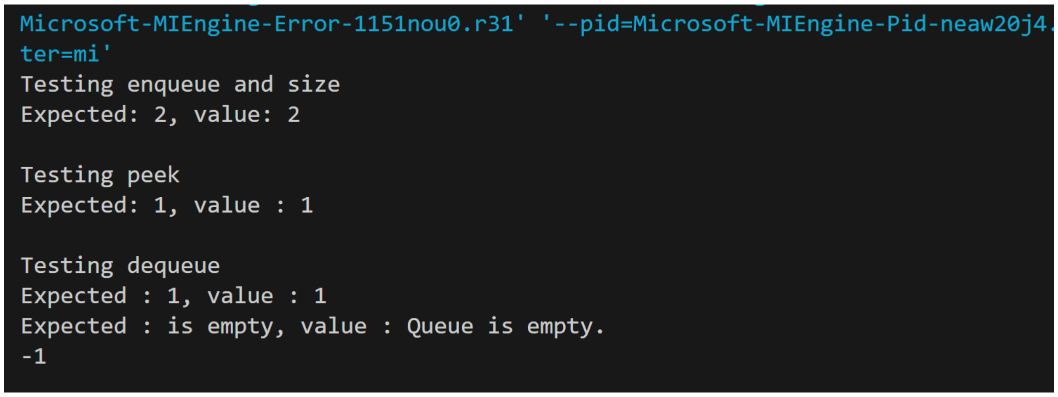
```
// looks at the front element
int peek() {
    // if empty return txt
    if (front == nullptr) {
        std::cout << "Queue is empty." << std::endl;
    }
    return front->data;
}
```

// test function for each of the major functions



```cpp
void testenQueueSize() {
    // makes que list
    lineQueue testsize;
    // test 1  adds two times to que
    std::cout << "Testing enqueue and size" << std::endl;
    testsize.enqueue(1);
    testsize.enqueue(2);
    // test size funtion
    std::cout << "Expected: 2, value: " << testsize.size() << std::endl;
    std::cout << std::endl;
}

void testPeek() {
    // makes que list
    lineQueue testpeek;
    // test 1  adds two times to que
    testpeek.enqueue(1);
    testpeek.enqueue(2);
    // tests the peek function
    std::cout << "Testing peek" << std::endl;
    std::cout << "Expected: 1, value : " << testpeek.peek() << std::endl;
    std::cout << std::endl;
}

void testDequeue() {
    // makes que list
    lineQueue testdequeue;
    // adds 1
    testdequeue.enqueue(1);
     // testing dequeue
    std::cout << "Testing dequeue" << std::endl;
    std::cout << "Expected : 1, value : " << testdequeue.dequeue() << std::endl;
    // runes a second time to check if its empty
    std::cout << "Expected : is empty, value : " << testdequeue.dequeue() << std::endl;
    std::cout << std::endl;
```

//results of test, worked as expected



```
Microsoft-MIEngine-Error-1151nou0.r31' '--pid=Microsoft-MIEngine-Pid-neaw20j4
ter=mi'
Testing enqueue and size
Expected: 2, value: 2

Testing peek
Expected: 1, value : 1

Testing dequeue
Expected : 1, value : 1
Expected : is empty, value : Queue is empty.
-1
```