**COVER PAGE**

Name: Maitra Patel (00984222)

Class: CIS 200 (Computer Science II)

Assignment: Project 2 -Part2 - Queues – Section 01 (Assigned – 3/7/2018)

Date: 3/18/2018

Semester: Winter 2018

1. **Problem Statement for Queue**
   1. Problem 1
      1. Implement a Class Queue of elements with string type. The queue must have makeEmpty, Dequeue, Enqueue, Print, Length, IsEmpty, and IsFull functions along with the copy constructor. Max size of this queue should be 100 elements. The queue must be implemented using linked lists.
2. **Lab Status**
   1. Problem 1: The program works for any type of string input into a queue. There are no issues with this program.
3. **Source Code:**

Compiler used for coding: Microsoft C/C++ -Visual Studio.

* 1. **Stacks:**
     1. **Source Code for header file of queues:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Queue of type string using linked list, and also have a copy constructer to copy one queue to another.

\*/

#ifndef QUEUE

#define QUEUE

struct NodeType {

string info;

NodeType \*next;

};

class Queue

{

private:

NodeType \*front; // It points to the front of a singly-linked list

NodeType \*rear; // It points to the end of a singly-linked list

public:

Queue(); // default constructor: Queue is created and empty

Queue(const Queue &x); // copy constructor: implicitly called

// for a deep copy

void MakeEmpty(); // Queue is made empty; you should deallocate all

// the nodes of the linked list

bool IsEmpty(); // test if the queue is empty

bool IsFull(); // test if the queue is full; assume MAXITEM=100

int length(); // return the number of elements in the queue

void Print(); // print the value of all elements in the queue in the sequence

// from the front to rear

void Enqueue(string x); // insert x to the rear of the queue

// Precondition: the queue is not full

void Dequeue(); // delete the element from the front of the queue

// Precondition: the queue is not empty

~Queue(); // Destructor: memory for the dynamic array needs to be deallocated

};

#endif !QUEUE

* + 1. **Source Code for implementation file of queues:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Queue of type string using linked list, and also have a copy constructer to copy one queue to another.

\*/

#include <iostream>

#include <string>

using namespace std;

#include "Queue.h"

//Constructer

Queue::Queue()

{

front = NULL;

rear = NULL;

}

//Copy Constructer

//Pre-Condtion: A queue to copy from

//Post-Condition: Queue is copied to this queue

Queue::Queue(const Queue &x)

{

//Make sure this queue is empty..

front = NULL;

rear = NULL;

NodeType \*tempPtr = x.front;

if (tempPtr == NULL)

{

front = NULL;

rear = NULL;

}

else

{

while (tempPtr != NULL)

{

Enqueue(tempPtr->info);

tempPtr = tempPtr->next;

}

}

}

//Make Empty Function

//Post-Condition: Empties the queue

void Queue::MakeEmpty()

{

string x = "0";

while ((front != NULL) && (rear != NULL))

{

Dequeue();

}

}

//IsEmpty function

//Post Condition: Returns if the function is empty or not

bool Queue::IsEmpty()

{

return ((rear == NULL) && (front == NULL));

}

//IsFull function

//Post Condition: Returns if the function is full or not

bool Queue::IsFull()

{

return(length() == 100);

}

//Enqueue function

//Post Condition: Adds a node to the end of queue

void Queue::Enqueue(string x)

{

NodeType \*newNode = new NodeType;

newNode->info = x;

newNode->next = NULL;

if (IsEmpty())

{

front = newNode;

rear = newNode;

}

else

{

NodeType \*currPtr = front;

while (currPtr->next != NULL)

{

currPtr = currPtr->next;

}

currPtr->next = newNode;

rear = newNode;

}

}

//Dequeue function

//Post Condition: Removes a node from the front of the queue

void Queue::Dequeue()

{

NodeType \*location;

if (IsEmpty())

{

cout << "Error: Cannot Dequeue, Queue is already Empty." << endl;

}

else

{

location = front;

if (front->next == NULL)

{

front = rear = NULL;

}

else

{

front = front->next;

}

delete location;

}

}

//Length function

//Post Condition: Returns the current size of queue

int Queue::length()

{

NodeType \*tempPtr = front;

int countNodes = 0;

while (tempPtr != NULL)

{

countNodes++;

tempPtr = tempPtr-> next;

}

return countNodes;

}

//Print function

//Post Condition: Prints the queue

void Queue::Print()

{

if (IsEmpty())

{

cout << "Queue is Empty!" << endl;

}

else

{

NodeType \*tempPtr = front;

while (tempPtr != NULL)

{

cout << tempPtr->info << " ";

tempPtr = tempPtr->next;

}

}

cout << endl;

}

//Deconstructer

//Post Condition: Deallocates the queue..

Queue::~Queue()

{

MakeEmpty();

delete front;

delete rear;

}

* + 1. **Source Code for main file of queues:**

/\*Author: Maitra Patel

\* Date Created: 3/10/2018

\* Date Modified: 3/18/2018

\* Purpose: Create a Queue of type string using linked list, and also have a copy constructer to copy one queue to another.

\*/

#include <string>

#include <iostream>

using namespace std;

#include "Queue.h"

int main()

{

Queue StringQueue;

string y;

StringQueue.MakeEmpty();

StringQueue.Dequeue();

StringQueue.Enqueue("John");

cout << "string length 3 = " << StringQueue.length() << endl;

StringQueue.Enqueue("Brown");

cout << "string length 4 = " << StringQueue.length() << endl;

StringQueue.Enqueue("Betty");

StringQueue.Dequeue();

cout << "The string queue contains : " << endl;

StringQueue.Print();

Queue StringQueue2 = StringQueue;

cout << "The string queue 2 contains: " << endl;

StringQueue2.Print();

StringQueue.MakeEmpty();

cout << "The string queue 3 contains: " << endl;

StringQueue.Print(); //FIXED ERROR: I'm assuming you meant to print StringQueue after making it empty..

return 0;

}

1. **Executable Module Instructions:**
   1. There are no specific module instructions required to run the code.
2. **Test Cases:**
   1. **Queues:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 1 | None  Note: Queue is empty | Message: Cannot Dequeue, Queue is empty | Message: Cannot Dequeue, Queue is empty | Pass |
| **Description:** To test the functionality of the Dequeue function with an empty queue | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 2 | None  Note: Enqueued once, then printed queue. Once again Enqueued and printed | Message: Length 1 = 1;  Length 2 = 2; | Message: Length 1 = 1;  Length 2 = 2; | Pass |
| **Description:** To test the functionality of the length function | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 3 | None  Note: Enqueued John, Brown, & Betty. Then Dequeued & printed. | Message:  Brown Betty | Message: Brown Betty | Pass |
| **Description:** To test the functionality of the Enqueue & Dequeue functions. | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 4 | None  Note: Queue 1 which contains Brown, & Betty is copied to Queue 2 | Queue2:  Brown Betty | Queue2: Brown Betty | Pass |
| **Description:** To test the functionality of the copy constructer. | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case#:** | **Inputs:** | **Expected Results:** | **Actual Results:** | **Pass/Fail:** |
| 5 | None  Note: Queue 1 is emptied & printed as Queue 3 | Queue3:  Queue is empty | Queue3: Queue is empty | Pass |
| **Description:** To test the functionality of the make empty function. | | | | |

1. **Running Logs/Outputs:**
2. **Running Log of Stacks in Screenshot format:**
   * **Test Case 1:**



The above image shows the functionality of Dequeue function which in this case is called when Queue is empty.

* + **Test Case 2:**



The above image shows the functionality of length function which in this case has been called twice. Once after Dequeuing from a size 3 queue & another after enqueuing into the queue.

* + **Test Case 3:**



The above image shows the functionality of the dequeue and enqueue functions in which “John” was dequeued & Betty was Enqueued.

* + **Test Case 4:**



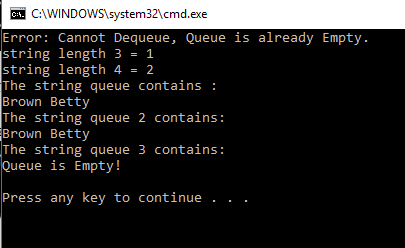
The above image shows the functionality of the copy constructer in which queue 1 was copied to queue 2 then printed.

* + **Test Case 5:**



The above image shows the functionality of the makeEmpty function in which Queue1 was emptied, then printed as Queue 3 for display.

* + **Screenshot of the overall program based on arguments called within main file:**



The above image shows the functionality of the overall program in which all functions were called in the main file for testing purposes.