PA01 - QueueList

Generated by Doxygen 1.8.6

Tue Jan 26 2016 13:33:52

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

3 Class Documentation

3.1 ListNode Struct Reference

Public Member Functions

- ListNode (int nodeData, ListNode ∗nextPtr)

  ListNode constructor.

Public Attributes

- int **dataItem**
- ListNode ∗ **next**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 ListNode::ListNode (int *nodeData,* ListNode ∗ *nextPtr*)

ListNode constructor.

Creates and initializes one list node with data and a next pointer

Precondition

ListNode uninitialized

Postcondition

ListNode initialized with data and next pointer

Algorithm

Initializes node with data via method initializers

Exceptions

None	No exceptional or error conditions handled in this method

Parameters

in	*nodeData*	provides the data to be stored in the node
in	*nextPtr*	provides the pointer address to be stored in the node

Note

Five spaces used before parameter parenthesis, seven spaces used before the initializer colon, three spaces used before the opening/closing curly braces

The documentation for this struct was generated from the following files:

- ListNode.h
- ListNode.cpp

3.2 QueueList Class Reference

Public Member Functions

- QueueList (int maxCapacity=10)

  Implementation of QueueList default constructor.
- QueueList (const QueueList &other)

  Implementation of QueueList copy constructor.
- ~QueueList ()

  Implementation of QueueList destructor.
- bool enqueue (int newDataItem)

  Implementation of method to add items to the QueueList.
- bool dequeue (int &newDataItem)

  Implementation of method to remove items from the QueueList.
- bool resetCapacity (int newCapacity)

  Implementation of method to change the capacity of the QueueList.
- void clear ()

  Implementation of method to clear all nodes in QueueList.
- QueueList & isAssigned (const QueueList &other)

  Implementation of method to copy the structure and data members of one QueueList to another.
- QueueList & operator= (const QueueList &other)

  Implementation of overloaded assignment operator.
- bool peekHead (int &dataVal) const

  Implementation of method to view the data item at the head of the QueueList object.
- void showStructure (char listID) const

  Implementation of method to print out the structure of a QueueList object to the screen.
- bool queueIsEmpty () const

  Implementation of method to check if the queue portion of a QueueList object is empty.
- bool queueIsFull () const

  Implementation of method to check whether the queue portion of a QueueList object is full.
- bool listIsEmpty () const

  Implementation of method to check for any nodes in the QueueList.

Static Public Attributes

- static const int **DISPLAY_WIDTH** = 5
- static const int **SMALL_STR_LEN** = 50
- static const char **SPACE** = ' '

Private Member Functions

- void displayChars (int numChars) const

  Implementation of char output method.

Private Attributes

- ListNode * **head**
- ListNode * **tail**
- int **capacityLimit**
- int **listCapacity**
- int **queueSize**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 QueueList::QueueList (int *maxCapacity* = `10`)

Implementation of QueueList default constructor.

Default constructor for QueueList class

Precondition

Uninitialized QueueList

Postcondition

QueueList object initialized with default values

Algorithm

Data members are assigned default values

Exceptions

None	

Parameters

`in`	*maxCapacity*	Data member capacityLimit initialized to maxCapacity which sets the maximum possible size of the QueueList (int)

Returns

None

Note

ListNode pointers are set to NULL, capacityLimit set to the parameter maxCapacity, and remaining data members set to zero

3.2.1.2 QueueList::QueueList (const QueueList & *other*)

Implementation of QueueList copy constructor.

Copy constructor for QueueList class

Precondition

Uninitialized QueueList

Postcondition

QueueList with same nodes and values as the QueueList passed in as a parameter

Algorithm

Data members of type ListNode pointer are initialized to NULL and the overloaded assignment operator is called on this

Exceptions

None	

Parameters

`in`	*other*	A const QueueList object passed in by reference to be copied by the calling object (QueueList)

Returns

None

Note

Data members of type ListNode pointer are initialized to NULL so that the QueueList will be empty when checked by other methods

3.2.1.3 QueueList::~QueueList ( )

Implementation of QueueList destructor.

Destructor for QueueList class

Precondition

QueueList object with or without nodes and data

Postcondition

All memory allocated to the QueueList object freed and data members set back to default values

Algorithm

The member method clear called to delete all nodes in the QueueList object

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.2.2 Member Function Documentation

3.2.2.1 void QueueList::clear ( )

Implementation of method to clear all nodes in QueueList.

If the QueueList has nodes then they are deleted and data members are set to default values

Precondition

An initialized QueueList object

Postcondition

The QueueList has all memory allocated to nodes freed and data members are set to default values

Algorithm

An if statement checks that there are nodes in the queue, then an event controlled loop goes through the queue with a temporary ListNode pointer and tail, moving from node to node and deleting all nodes and then data members are assigned default values

Exceptions

If	there are no nodes in the QueueList then the method exits

Parameters

None	

Returns

None

Note

The data member capacityLimit is not set to default its default value as this value should persist for subsequent usage of the data structure

3.2.2.2 bool QueueList::dequeue (int & *newDataItem*)

Implementation of method to remove items from the QueueList.

If there are items to remove from the queue then the item at head is removed and the parameter passed into the method is assigned the value of the item being removed

Precondition

A QueueList object which has been initialized

Postcondition

If the QueueList contained at least one item then it is removed from the head of the queue and head is moved

Algorithm

An if statement checks whether or not the queue is empty and if so then false is returned, otherwise the parameter passed in is assigned the value of the data member dataItem at the head node and head is moved to the next node and queueSize is decremented

Exceptions

None	

Parameters

out	*newDataItem*	An int passed in by reference which will accept the value of the item removed from the head of the queue (int)

Returns

A bool is returned corresponding to whether or not the item was successfully removed from the queue (bool)

Note

None

3.2.2.3 void QueueList::displayChars (int *numChars*) const [private]

Implementation of char output method.

An int is taken in as a parameter which corresponds to the number of the char SPACE to be output to the screen

Precondition

A positive, non-zero int is entered as a parameter

Postcondition

The method will output spaces to the screen in the amount of the int passed in as a parameter

Algorithm

An if statement checks if numChars is greater than zero and a counter controlled loop with numChars as the controlling value outputs SPACE, a global constant corresponding to a space, to the screen

Exceptions

If	numChars is zero or a negative number then the method exits

Parameters

in	*numChars*	An int greater than zero which corresponds to the number of spaces output to the screen (int)

Returns

None

Note

SPACE is a global constant char which is assigned the value of a single space

3.2.2.4 bool QueueList::enqueue (int *newDataItem*)

Implementation of method to add items to the QueueList.

The int parameter passed into the method is added to the queue after tail if there is space on the list

Precondition

A QueueList object which has been initialized

Postcondition

If there is in room in the QueueList then the int parameter passed into the method is added to the queue, another node having been created for the item if necessary and tail is moved

Algorithm

An if statement checks that the queue is not full and whether capacityLimit is greater than zero, if not false is returned, if so if else statements check 1) whether or not there are any nodes and if not one is created for the item and assigned to head and tail, 2) whether queueSize and listCapacity are equal in which case a new node is created for the item and linked with tail, and 3) if queueSize is less than listCapacity in which case if queueSize is zero then the item is assigned to the node at head and if not then tail is moved to the next node and the item is added to that node

Exceptions

None	

Parameters

`in`	*newDataItem*	An int which, if there is space in the queue, is added to to the dataItem portion of the node following tail (int)

Returns

A bool is returned corresponding to whether or not the item was successfully added to the queue (bool)

Note

Data members correponding the size of the queue and the list are incremented

3.2.2.5 QueueList & QueueList::isAssigned (const QueueList & *other*)

Implementation of method to copy the structure and data members of one QueueList to another.

The structure and data members of the QueueList object passed in by reference is copied to the calling QueueList object

Precondition

A QueueList object

Postcondition

A QueueList object containing all nodes and data members copied from the QueueList object passed in as a parameter

Algorithm

The method clear is called to delete all nodes and reset default values, then the values of the data members from the QueueList object passed in as a parameter are assigned to those of the calling object, and an event controlled loop moves a temporary node pointer through the queue to be copied while another creates nodes for the calling object and the data items are copied into them

Exceptions

If	the calling QueueList object and the QueueList object passed in as a parameter are the same object then the method exits

Parameters

`in`	*other*	A QueueList object passed in by reference which will have its structure and data members copied to the calling obejct (QueueList)

Returns

The calling QueueList object is returned by this dereferenced (QueueList)

Note

None

3.2.2.6 bool QueueList::listIsEmpty ( ) const

Implementation of method to check for any nodes in the QueueList.

The QueueList object is checked for any nodes, used or unused, and returns a corresponding bool value

Precondition

An initialized QueueList object

Postcondition

True is returned if the QueueList is empty, otherwise false is returned

Algorithm

An if statement checks whether the data members of type ListNode pointer are NULL

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not the QueueList is empty (bool)

Note

None

3.2.2.7 QueueList & QueueList::operator= (const QueueList & *other*)

Implementation of overloaded assignment operator.

The method isAssigned is called to copy the QueueList passed in as a parameter into the calling QueueList object

Precondition

A QueueList object

Postcondition

A QueueList object containing all nodes and data members copied from the QueueList object passed in as a parameter

Algorithm

The method isAssigned is called on the parameter passed in and its result is returned

Exceptions

None	

Parameters

`in`	*other*	A QueueList object passed in by reference which will have its structure and data members copied to the calling obejct (QueueList)

Returns

The result of the call to the method isAssigned is returned (QueueList)

Note

None

3.2.2.8 bool QueueList::peekHead (int & *dataVal*) const

Implementation of method to view the data item at the head of the QueueList object.

The data item at the head of the queue is assigned the parameter passed in by reference and a bool corresponding to whether or not it was successful is returned

Precondition

A initialized QueueList

Postcondition

True is returned if the queue is not empty and the parameter passed in will contain the value of the data member in the node at head, false is returned if there are no used nodes

Algorithm

An if statement checks if the queue is empty and if it's not then the value of the data member dataItem at the head node is assigned to the parameter passed into the method

Exceptions

None	

Parameters

out	*dataVal*	An int passed in by reference which will accept the value of the data member of the node at head (int)

Returns

A bool corresponding to whether or not the method was successful (bool)

Note

None

3.2.2.9 bool QueueList::queueIsEmpty ( ) const

Implementation of method to check if the queue portion of a QueueList object is empty.

The QueueList object is checked for any used nodes and returns a corresponding bool value

Precondition

An initialized QueueList object

Postcondition

True is returned if there are no used nodes in the QueueList, otherwise false is returned

Algorithm

With a call to the member method listIsEmpty, an if statement checks if the list is empty or if the data member queueSize is zero

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not there are any used nodes in the QueueList (bool)

Note

None

3.2.2.10 bool QueueList::queueIsFull ( ) const

Implementation of method to check whether the queue portion of a QueueList object is full.

The current size of the queue, the used nodes, is checked against the total capacity of the QueueList and a corresponding bool is returned

Precondition

An initialized QueueList object

Postcondition

True is returned if the queue is full, otherwise false is returned

Algorithm

An if statement checks if the data member queueSize is equal to capacityLimit

Exceptions

None	

Parameters

None	

Returns

A bool is returned corresponding to whether or not the queue portion of the QueueList is full (bool)

Note

None

3.2.2.11 bool QueueList::resetCapacity (int *newCapacity*)

Implementation of method to change the capacity of the QueueList.

The maximum number of allowed nodes, capacityLimit, is changed to the parameter passed in if it's greater than the size of the queue

Precondition

An initialized QueueList object

Postcondition

The QueueList object has the number of possible nodes, limitCapacity, changed and unused nodes are deleted if necessary

Algorithm

An int counter is set equal to listCapacity minus the parameter passed in, newCapacity, which will be the number of nodes to be deleted if necessary, then if statements check that 1) newCapacity is greater than capacityLimit in which case capacityLimit is assigned the value of newCapacity and true is returned, 2) newCapacity is less than listCapacity and greater than or equal to queueSize in which case if newCapacity is zero then clear is called, otherwise tail and/or head are moved depending on whether or not newCapacity is equal to one and a counter controlled loop moves a node pointer through the list after tail and deletes nodes and true is returned, otherwise if neither of those main branches are entered then false is returned

Exceptions

None	

Parameters

`in`	*newCapacity*	An int that specifies what the capacityLimit, or the maximum total number of allowed nodes, will be changed to (int)

Returns

A bool is returned corresponding to whether or not the resize was successful (bool)

Note

None

3.2.2.12 void QueueList::showStructure (char *listID*) const

Implementation of method to print out the structure of a QueueList object to the screen.

QueueList data members are printed to the screen along with the data items of the nodes in the list in rows of five

Precondition

A QueueList object

Postcondition

The QueueList nodes printed to the screen

Algorithm

The cstdlib function sprintf creates a string, buffer, containing identifying information, then an if statement checks if the list is empty, if so an indication of that is printed to the screen, if not then a counter controlled loop goes through the list with a node pointer and prints each node's data item to the screen preceded by a char which indicates if it's the head, tail, in in the queue otherwise or unused, and surrounded by brackets

Exceptions

None	

Parameters

`in`	*listID*	A char which represents an identifier of the QueueList to be printed to the screen

Returns

None

Note

None

The documentation for this class was generated from the following files:

- QueueList.h
- QueueList.cpp

4 File Documentation

4.1 ListNode.cpp File Reference

Implementation file for ListNode class (struct)

```
#include "ListNode.h"
```

4.1.1 Detailed Description

Implementation file for ListNode class (struct) Implements constructor for ListNode class

Version

1.00 Michael Leverington (30 August 2015) Original development of class

Note

Depends on ListNode.h

4.2 ListNode.h File Reference

Header file for list node.

Classes

- struct ListNode

4.2.1 Detailed Description

Header file for list node. Basic list node header file incorporates struct with integer data item and pointer to next node

Version

1.00 Michael Leverington (30 August 2015) Original development of class

Note

Materials loosely base on Linked List exercise in Brandl, et. al. in C++ Data Structures: A Laboratory Course, (c) 2009.

4.3 PA01.cpp File Reference

Driver program to exercise QueueList and ListNode classes.

```
#include <iostream>
#include <cstring>
#include "QueueList.h"
```

Functions

- void ShowMenu ()

  Displays choice of commands for exercising linked list.
- char GetCommandInput ()

  Acquires command input from user.
- int GetDataInput (char inputChar)

  Acquires data input if a specific command was input.
- int **main** ()

Variables

- const int **SMALL_STR_LEN** = 25
- const bool **VERBOSE** = true
- const char **ENDLINE_CHAR** = '\n'

4.3.1 Detailed Description

Driver program to exercise QueueList and ListNode classes. Allows for testing all QueueList methods in an interactive operation

Version

1.00 Original development (13 January 2016)

Note

Requires ListNode.h, ListNode.cpp, QueueList.h, QueueList.cpp

4.3.2 Function Documentation

4.3.2.1 char GetCommandInput ( )

Acquires command input from user.

Command letters are unique combinations of three letters

Precondition

None

Postcondition

Three characters are placed into a character array

Algorithm

Characters are input to array elements one at a time

Exceptions

None	

Parameters

None	

Returns

Input character captured via iostream

Note

None

4.3.2.2 int GetDataInput (char *inputChar*)

Acquires data input if a specific command was input.

Accepts input for all operations starting with "e" or "r"; does not input anything otherwise

Precondition

None

Postcondition

Value is input and returned if control requires it, otherwise, zero is returned

Algorithm

Tests for character, accepts input if appropriate, returns data value if input accepted, or zero if no input accepted

Exceptions

None	

Parameters

in	*inputChar*	Character is used to check for input control

Returns

Data value accepted or zero (int)

Note

None

4.3.2.3 void ShowMenu ( )

Displays choice of commands for exercising linked list.

Command letters displayed are unique characters specified as shown

Precondition

None

Postcondition

Choice of commands is displayed as specified

Algorithm

Standard output operations for each command line available

Exceptions

None	

Parameters

None	

Returns

None

Note

Five spaces for parameter parentheses, three spaces for curly braces

4.4 QueueList.cpp File Reference

Implementation file for QueueList class.

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include "QueueList.h"
```

Variables

- static const int **ONE** = 1
- static const int **ZERO** = 0

4.4.1 Detailed Description

Implementation file for QueueList class. Implements member methods of QueueList class

Version

1.00 Bryan Kline (25 January 2016)

Note

Requires QueueList.h

4.5 QueueList.h File Reference

Header file for queue implementation using linked list.

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include "ListNode.h"
```

Classes

- class QueueList

4.5.1 Detailed Description

Header file for queue implementation using linked list. Definitions of all members to be used in the QueueList class

Version

1.00 Michael Leverington(13 January 2016)

Note

None
Depends on ListNode header file

Index