CS 302 Project 5

Generated by Doxygen 1.8.6

Tue Mar 24 2015 16:04:25

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 arrayQueue< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- arrayQueue (int)
- ~arrayQueue ()
- bool isEmpty () const
- bool enqueue (const ItemType &newEntry)
- bool dequeue ()
- ItemType peek () const

3.1.1 Detailed Description

template<class ItemType>class arrayQueue< ItemType >

Array based queue. Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.1.2 Constructor & Destructor Documentation

3.1.2.1 template<class ItemType > arrayQueue< ItemType >::arrayQueue (int *val*)

arrayQueue constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Sufficient dynamic memory available to create an ItemType array of size == val

Postcondition

arrayQueue is created and ready for first entry.

Returns

None.

3.1.2.2 template<class ItemType > arrayQueue< ItemType >::~arrayQueue ( )

arrayQueue destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the arrayQueue object has been freed.

Returns

None.

3.1.3 Member Function Documentation

3.1.3.1 template<class ItemType > bool arrayQueue< ItemType >::dequeue ( )

arrayQueue dequeue function. Attempts to remove an item from the front of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value removed from the queue, provided the queue was not empty.

Returns

bool True if the dequeue was successful, false if it failed.

3.1.3.2 template<class ItemType > bool arrayQueue< ItemType >::enqueue (const ItemType & *newEntry*)

arrayQueue enqueue function. Attempts to enqueue an item at the back of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value added at the back of the queue if the queue was not full.

Parameters

newEntry	The value to be enqueued.

Returns

bool True if the enqueue was successful, false if it failed.

3.1.3.3 template<class ItemType > bool arrayQueue< ItemType >::isEmpty ( ) const

arrayQueue isEmpty function. Checks whether the queue is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.1.3.4 template<class ItemType > ItemType arrayQueue< ItemType >::peek ( ) const

arrayQueue peek function. Returns the item currently stored at the front of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

The queue must not be empty. Caveat emptor!

Postcondition

Item at the front of the queue returned; queue contents unchanged.

Returns

ItemType The item returned by the peek operation.

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

- queues.h
- queue.cpp

3.2 arraySortedList< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- arraySortedList (int)
- ~arraySortedList ()
- bool insertSorted (const ItemType &newEntry)
- bool removeSorted (const ItemType &anEntry)
- int getPosition (const ItemType &newEntry) const
- bool isEmpty () const
- int getLength () const
- bool remove (int position)
- void clear ()
- ItemType getEntry (int position) const

3.2.1 Detailed Description

template<class ItemType>class arraySortedList< ItemType >

Array based sorted list. Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.2.2 Constructor & Destructor Documentation

3.2.2.1 template<class ItemType > arraySortedList< ItemType >::arraySortedList (int *val*)

arraySortedList constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Sufficient dynamic memory available to create an ItemType array of size == val

Postcondition

arraySortedList is created and ready for first entry.

Returns

None.

3.2.2.2 template<class ItemType > arraySortedList< ItemType >::~arraySortedList ( )

arraySortedList destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the arraySortedList object has been freed.

Returns

None.

3.2.3 Member Function Documentation

3.2.3.1 template<class ItemType > void arraySortedList< ItemType >::clear ( )

arraySortedList clear function. Resets the sorted list to an empty state. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Array sorted list is reset to its empty state.

Returns

None.

3.2.3.2 template<class ItemType > ItemType arraySortedList< ItemType >::getEntry (int *position*) const

arraySortedList getEntry function. Returns the item in the sorted list located at the specified location. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

The list must not be empty, and the position requested must not be greated than the length of the list. Caveat emptor!

Postcondition

The value in the sorted list at target position returned; list contents unchanged.

Parameters

position	The location in the sorted list from which to retrieve an item.

Returns

ItemType The item at the specified location.

3.2.3.3 template<class ItemType > int arraySortedList< ItemType >::getLength ( ) const

arraySortedList getLength function. Returns the number of items, ie the length, of the sorted list. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; length value returned.

Returns

int The length of the sorted list.

3.2.3.4 template<class ItemType > int arraySortedList< ItemType >::getPosition (const ItemType & *newEntry*) const

arraySortedList getPosition function. Searches the sorted list for a value matching the one provided, and returns its index if it is found. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; position of a value in the sorted list that == newEntry returned if one was found. If not found, returns -1

Parameters

newEntry	The entry which is to be attempted to be located in the sorted list.

Returns

int The position of the value if found, -1 if not found.

3.2.3.5 template<class ItemType > bool arraySortedList< ItemType >::insertSorted (const ItemType & *newEntry*)

arraySortedList insertSorted function. Inserts an item into the sorted list in a position that will maintain the list's sorted property (ascending order). !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

newEntry inserted into the list if it was not full, list unchanged otherwise

Parameters

newEntry	The item to be added into the sorted list.

Returns

bool True if insert operation was successful, false otherwise

3.2.3.6 template<class ItemType > bool arraySortedList< ItemType >::isEmpty ( ) const

arraySortedList isEmpty function. Checks whether the sorted list is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.2.3.7 template<class ItemType > bool arraySortedList< ItemType >::remove (int *position*)

arraySortedList remove function. Removes the item at a specified location from the sorted list while maintaining maintain the list's sorted property (ascending order). !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value at position removed, so long as the list was not empty and the requested position was not > the length of the list.

Parameters

position	The position of the list from which to remove a value.

Returns

bool True if removal operation was successful, false otherwise.

3.2.3.8 template<class ItemType > bool arraySortedList< ItemType >::removeSorted (const ItemType & *anEntry*)

arraySortedList removeSorted function. Removes a specified item from the sorted list while maintaining maintain the list's sorted property (ascending order). !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

If the value was located in the list, it has been removed. If the array was empty or the value was not found, the list is unchanged.

Parameters

anEntry	The entry to be removed from the sorted list.

Returns

bool True if remove operation was successful, false otherwise.

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

- queues.h
- priorityQueue.cpp

3.3 event Class Reference

`#include <event.h>`

Public Member Functions

- event ()
- void setInitialTime (const int &)
- int getInitialTime () const
- void setArrivalOrDeparture (const char &)
- char getArrivalOrDeparture () const
- void setTransactionDuration (const int &)
- int getTransactionDuration () const
- bool operator< (const event &) const
- bool operator<= (const event &) const
- bool operator> (const event &) const
- bool operator>= (const event &) const
- bool operator== (const event &) const
- bool operator!= (const event &) const

3.3.1 Detailed Description

Event class for use in queue and priority queue simulation operations. Uses char 'a' in arrivalOrDeparture member to signify arrival and char 'd' to signify departure. TransactionDuration member is used in the departure event in sims with multiple lines/tellers to denote the line/teller of origin for the event. Uses overloaded comparison operators in order to ensure correct sorted list operations.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 event::event ( )

Event class constructor.

Precondition

None.

Postcondition

Event is created with default values of 0, 0, 'a'

Returns

None.

3.3.3 Member Function Documentation

3.3.3.1 char event::getArrivalOrDeparture ( ) const

Event getArrivalOrDeparture function. Returns the arrivalOrDeparture data member of an event object.

Precondition

None.

Postcondition

Event object's arrivalOrDeparture returned; object itself unchanged.

Returns

char - The value held by arrivalOrDeparture.

3.3.3.2 int event::getInitialTime ( ) const

Event getInitialTime function. Returns the initialTime data member of an event object.

Precondition

None.

Postcondition

Event object's initialTime returned; object itself unchanged.

Returns

int - The value held by initialTime.

3.3.3.3 int event::getTransactionDuration ( ) const

Event getTransactionDuration function. Returns the transactionDuration data member of an event object.

Precondition

None.

Postcondition

Event object's transactionDuration returned; object itself unchanged.

Returns

int - The value held by transactionDuration.

3.3.3.4 bool event::operator!= (const event & *val*) const

Event overloaded != operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime != val's initialTime, false otherwise.

3.3.3.5 bool event::operator< (const event & *val*) const

Event overloaded $<$ operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime $<$ val's initialTime, false otherwise.

3.3.3.6 bool event::operator<= (const event & *val*) const

Event overloaded <= operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime <= val's initialTime, false otherwise.

3.3.3.7 bool event::operator== (const event & *val*) const

Event overloaded == operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime == val's initialTime, false otherwise.

3.3.3.8 bool event::operator> (const event & *val*) const

Event overloaded > operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime > val's initialTime, false otherwise.

3.3.3.9 bool event::operator>= (const event & *val*) const

Event overloaded >= operator. Evaluates using initialTime data members of the objects.

Precondition

None.

Postcondition

Bool returned; both objects unchanged.

Parameters

val	The event to which this one is being compared.

Returns

bool - True if this object's initialTime >= val's initialTime, false otherwise.

3.3.3.10 void event::setArrivalOrDeparture (const char & *val*)

Event setArrivalOrDeparture function. Set for arrivalOrDeparture data member.

Precondition

None.

Postcondition

arrivalOrDeparture data member holds val.

Parameters

val	The char value to which arrivalOrDeparture is to be set.

Returns

None.

3.3.3.11 void event::setInitialTime (const int & *val*)

Event setInitialTime function. Set for initialTime data member.

Precondition

None.

Postcondition

initialTime data member holds val.

Parameters

val	The int value to which initialTime is to be set.

Returns

None.

3.3.3.12 void event::setTransactionDuration (const int & *val*)

Event setTransactionDuration function. Set for transactionDuration data member.

Precondition

None.

Postcondition

transactionDuration data member holds val.

Parameters

val	The int value to which transactionDuration is to be set.

Returns

None.

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

- event.h
- event.cpp

3.4 linkedQueue< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- linkedQueue ()
- ~linkedQueue ()
- bool isEmpty () const
- bool enqueue (const ItemType &newEntry)
- bool dequeue ()
- ItemType peek () const

3.4.1 Detailed Description

template<class ItemType>class linkedQueue< ItemType >

Node based queue. Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.4.2 Constructor & Destructor Documentation

3.4.2.1 template<class ItemType > linkedQueue< ItemType >::linkedQueue ( )

linkedQueue constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

linkedQueue is created and ready for first entry.

Returns

None.

3.4.2.2 template<class ItemType > linkedQueue< ItemType >::~linkedQueue ( )

linkedQueue destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the arrayQueue object has been freed.

Returns

None.

3.4.3 Member Function Documentation

3.4.3.1 template<class ItemType > bool linkedQueue< ItemType >::dequeue ( )

linkedQueue dequeue function. Attempts to remove an item from the front of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value removed from the queue, provided the queue was not empty.

Returns

bool True if the dequeue was successful, false if it failed.

3.4.3.2 template<class ItemType > bool linkedQueue< ItemType >::enqueue (const ItemType & *newEntry*)

linkedQueue enqueue function. Attempts to enqueue an item at the back of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value added at the back of the queue.

Parameters

newEntry	The value to be enqueued.

Returns

bool Always returns true.

3.4.3.3 template<class ItemType > bool linkedQueue< ItemType >::isEmpty ( ) const

linkedQueue isEmpty function. Checks whether the sorted list is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.4.3.4 template<class ItemType > ItemType linkedQueue< ItemType >::peek ( ) const

linkedQueue peek function. Returns the item currently stored at the front of the queue. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

The queue must not be empty. Caveat emptor!

Postcondition

Item at the front of the queue returned; queue contents unchanged.

Returns

ItemType The item returned by the peek operation.

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

- queues.h
- queue.cpp

3.5 linkedSortedList< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- linkedSortedList ()
- ~linkedSortedList ()
- void insertSorted (const ItemType &newEntry)
- bool remove (int position)
- int getPosition (const ItemType &newEntry) const
- bool isEmpty () const
- int getLength () const
- void clear ()
- ItemType getEntry (int position)

3.5.1 Detailed Description

template<class ItemType>class linkedSortedList< ItemType >

Node based sorted list. Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.5.2 Constructor & Destructor Documentation

3.5.2.1 template<class ItemType > linkedSortedList< ItemType >::linkedSortedList ( )

linkedSortedList constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

linkedSortedList is created and ready for first entry.

Returns

None.

3.5.2.2 template<class ItemType > linkedSortedList< ItemType >::~linkedSortedList ( )

arraySortedList destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the sorted list, if any, has been freed

Returns

None.

3.5.3 Member Function Documentation

3.5.3.1 template<class ItemType > void linkedSortedList< ItemType >::clear ( )

linkedSortedList clear function. Resets the sorted list to an empty state. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List is reset to its empty state. Any dynamic memory used has been freed.

Returns

None.

3.5.3.2 template<class ItemType > ItemType linkedSortedList< ItemType >::getEntry (int *position*)

linkedSortedList getEntry function. Returns the item in the sorted list located at the specified location. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

The list must not be empty, and the position requested must not be greater than the length of the list. Caveat emptor!

Postcondition

The value in the sorted list at target position returned; list contents unchanged.

Parameters

position	The location in the sorted list from which to retrieve an item.

Returns

ItemType The item at the specified location.

3.5.3.3 template<class ItemType > int linkedSortedList< ItemType >::getLength ( ) const

linkedSortedList getLength function. Returns the number of items, ie the length, of the sorted list. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; length value returned.

Returns

int The length of the sorted list.

3.5.3.4 template<class ItemType > int linkedSortedList< ItemType >::getPosition (const ItemType & *newEntry*) const

3.5.3.5 template<class ItemType > void linkedSortedList< ItemType >::insertSorted (const ItemType & *newEntry*)

linkedSortedList insertSorted function. Inserts an item into the sorted list in a position that will maintain the list's sorted property (ascending order). !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

newEntry inserted into the list in sorted position.

Parameters

newEntry	The item to be added into the sorted list.

Returns

None.

3.5.3.6 template<class ItemType > bool linkedSortedList< ItemType >::isEmpty ( ) const

linkedSortedList isEmpty function. Checks whether the sorted list is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

List unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.5.3.7 template<class ItemType > bool linkedSortedList< ItemType >::remove (int *position*)

linkedSortedList removeSorted function. Removes a specified item from the sorted list while maintaining maintain the list's sorted property (ascending order). !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

If the value was located in the list, it has been removed. If the array was empty or the value was not found, the list is unchanged.

Parameters

anEntry	The entry to be removed from the sorted list.

Returns

bool True if remove operation was successful, false otherwise.

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

- queues.h
- priorityQueue.cpp

3.6 node< ItemType > Class Template Reference

`#include <node.h>`

Public Member Functions

- node ()
- node (const ItemType &anItem)
- node (const ItemType &anItem, node< ItemType > *nextnodePtr)
- void setItem (const ItemType &anItem)
- void setNext (node< ItemType > *nextnodePtr)
- ItemType getItem () const
- node< ItemType > * getNext () const

3.6.1 Detailed Description

template<class ItemType>class node< ItemType >

Templated node class for use in linked-list based queue/sortedlist/priority queue operations in this program. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.6.2 Constructor & Destructor Documentation

3.6.2.1 template<class ItemType > node< ItemType >::node ( )

Node default constructor. Sets next value to nullptr. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Enough available memory for a new node to be created.

Postcondition

Node created with next == nullptr.

Returns

None.

3.6.2.2 template<class ItemType > node< ItemType >::node (const ItemType & *anItem*)

Node parameterized constructor. Sets next value to nullptr and item to the one provided. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Enough available memory for a new node to be created.

Postcondition

Node created with next == nullptr and item == anItem.

Parameters

anItem	The item which will be held by this node.

Returns

None.

3.6.2.3 template<class ItemType > node< ItemType >::node (const ItemType & *anItem,* node< ItemType > * *nextNodePtr*)

Node parameterized constructor. Sets next value to nextNodePtr and item to the one provided. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Enough available memory for a new node to be created.

Postcondition

Node created with next == nextNodePtr and item == anItem

Parameters

anItem	The item which will be held by this node
nextNodePtr	A pointer which will be held by the node's next data member.

Returns

None.

3.6.3 Member Function Documentation

3.6.3.1 template<class ItemType > ItemType node< ItemType >::getItem ( ) const

Node getItem function. Returns the item held in this node's item data member. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value returned; node is unchanged

Returns

ItemType The item held by the node's item data member.

3.6.3.2 template<class ItemType > node< ItemType > * node< ItemType >::getNext ( ) const

Node getNext function. Returns the pointer held in this node's next data member. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Value returned; node is unchanged

Returns

node<ItemType>* The pointer held by the node's next data member.

3.6.3.3 template<class ItemType > void node< ItemType >::setItem (const ItemType & *anItem*)

Node setItem function. Sets the node's item data member to the one provided. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Node's item data member set to anItem

Parameters

anItem	The item which will be held by this node.

Returns

None.

3.6.3.4 template<class ItemType > void node< ItemType >::setNext (node< ItemType > * *nextNodePtr*)

Node setNext function. Sets the node's next data member to the one provided. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Node's next data member set to nextNodePtr

Parameters

nextNodePtr	The pointer which this node's next data member will point to.

Returns

None.

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

- node.h
- node.cpp

3.7 priorityQueueArray< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- priorityQueueArray (int)
- ~priorityQueueArray ()
- bool isEmpty () const
- bool add (const ItemType &newEntry)
- bool remove ()
- ItemType peek () const

3.7.1 Detailed Description

template<class ItemType>class priorityQueueArray< ItemType >

Array based priority queue (using arraySortedList). Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.7.2 Constructor & Destructor Documentation

3.7.2.1 template<class ItemType > priorityQueueArray< ItemType >::priorityQueueArray (int *val*)

priorityQueueArray constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Sufficient dynamic memory available to create an ItemType array of size == val

Postcondition

Priority queue created.

Returns

None.

3.7.2.2 template<class ItemType > priorityQueueArray< ItemType >::~priorityQueueArray ( )

priorityQueueArray destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the priorityQueueArray object has been freed.

Returns

None.

3.7.3 Member Function Documentation

3.7.3.1 template<class ItemType > bool priorityQueueArray< ItemType >::add (const ItemType & *newEntry*)

priorityQueueArray add function. Attempts to add an item to the priority queue while maintaining the queue's arrangement in order of priority.

Precondition

None.

Postcondition

newEntry added to PQ if the PQ was not already full.

Parameters

newEntry	The value to be added to the PQ.

Returns

bool True if the add was successful, false otherwise.

3.7.3.2 template<class ItemType > bool priorityQueueArray< ItemType >::isEmpty ( ) const

priorityQueueArray isEmpty function. Checks whether the priority queue is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

PQ unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.7.3.3 template<class ItemType > ItemType priorityQueueArray< ItemType >::peek ( ) const

priorityQueueArray peek function. Returns the highest priority item currently stored in the PQ.

Precondition

The PQ must not be empty. Caveat emptor!

Postcondition

Highest priority item returned; PQ contents unchanged.

Returns

ItemType The highest priority item in the PQ.

3.7.3.4 template<class ItemType > bool priorityQueueArray< ItemType >::remove ( )

priorityQueueArray remove function. Attempts to remove the highest priority item from the PQ.

Precondition

None.

Postcondition

Highest priority item removed from the PQ if the PQ was not empty.

Returns

bool True if the removal was successful, false otherwise.

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

- queues.h
- priorityQueue.cpp

3.8 priorityQueueLinked< ItemType > Class Template Reference

`#include <queues.h>`

Public Member Functions

- priorityQueueLinked ()
- ~priorityQueueLinked ()
- bool isEmpty () const
- bool add (const ItemType &newEntry)
- bool remove ()
- ItemType peek () const

3.8.1 Detailed Description

template<class ItemType>class priorityQueueLinked< ItemType >

Node based priority queue (using linkedSortedList). Templated. !!! ADAPTED FROM TEXTBOOK CODE !!!

3.8.2 Constructor & Destructor Documentation

3.8.2.1 template<class ItemType > priorityQueueLinked< ItemType >::priorityQueueLinked ( )

priorityQueueLinked constructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Sufficient dynamic memory available to create a linkedSortedList object.

Postcondition

Priority queue created.

Returns

None.

3.8.2.2 template<class ItemType > priorityQueueLinked< ItemType >::~priorityQueueLinked ( )

priorityQueueLinked destructor. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

Dynamic memory allocated for the priorityQueueLinked object has been freed.

Returns

None.

3.8.3 Member Function Documentation

3.8.3.1 template<class ItemType > bool priorityQueueLinked< ItemType >::add (const ItemType & *newEntry*)

priorityQueueLinked add function. Adds an item to the priority queue while maintaining the queue's arrangement in order of priority.

Precondition

Sufficient dynamic memory available for a new ItemType object.

Postcondition

newEntry added to the PQ.

Parameters

newEntry	The value to be added to the PQ.

Returns

bool Always returns true.

3.8.3.2 template<class ItemType > bool priorityQueueLinked< ItemType >::isEmpty ( ) const

priorityQueueLinked isEmpty function. Checks whether the priority queue is empty. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

None.

Postcondition

PQ unchanged; bool signifying empty status returned.

Returns

bool True if the list is empty, false otherwise

3.8.3.3 template<class ItemType > ItemType priorityQueueLinked< ItemType >::peek ( ) const

priorityQueueLinked peek function. Returns the highest priority item currently stored in the PQ.

Precondition

The PQ must not be empty. Caveat emptor!

Postcondition

Highest priority item returned; PQ contents unchanged.

Returns

ItemType The highest priority item in the PQ.

3.8.3.4 template<class ItemType > bool priorityQueueLinked< ItemType >::remove ( )

priorityQueueLinked remove function. Attempts to remove the highest priority item from the PQ.

Precondition

None.

Postcondition

Highest priority item removed from the PQ if the PQ was not empty.

Returns

bool True if the removal was successful, false otherwise.

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

- queues.h
- priorityQueue.cpp

3.9 simA Class Reference

`#include <simA.h>`

Public Member Functions

- void simOne (const bool &, const int &)
- void simTwo (const bool &, const int &)
- void simThree (const bool &, const int &)
- void outputToConsole (const int &, const int &) const
- void initCSVOutput ()
- void outputToCSV (const int &, const int &)
- void endCSVOutput ()
- void resetStats ()

3.9.1 Detailed Description

Sim class using array based implementation for queue and priority queue data structures.. Conducts bank queue simulation operations as specified in the prompt. Uses simulation object data member to track statistics of the operation and conduct console and file output. User can call three different simulations of specified size, with full step by step of each event printed to console or summary statistic data. Pretty nifty. Supported sizes are 10, 1 000, 10 000, 100 000, 1 000 000.

3.9.2 Member Function Documentation

3.9.2.1 void simA::endCSVOutput ( )

Sim class endCSVOutput function. Calls the endCSVOutput function of the sim class's stats data member. Closes output.csv file, ending file output operations.

Precondition

None.

Postcondition

Fstream to output.csv closed.

Returns

None.

3.9.2.2 void simA::initCSVOutput ( )

Sim initCSVOutput function. Calls the initCSVOutput function of the sim class's stats data member. Opens output file (output.csv) for subsequent functions and prints a header row to the file.

Precondition

None.

Postcondition

Fstream to output.csv opened, header row printed in output.csv

Returns

None.

3.9.2.3 void simA::outputToConsole (const int & *simNum,* const int & *size*) const

Sim outputToConsole function. Calls the outputToConsole function of the sim class's stats data member. Spits processed statistics data to console. Supported sizes noted below.

Precondition

None.

Postcondition

Processed values printed to console; stored values unchanged.

Parameters

simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.
size	The size of the simulation that was run. Supported sizes for this function are 10, 1 000, 10 000, 100 000, 1 000 000. For size 10 and 1 000 000, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.9.2.4 void simA::outputToCSV (const int & *simNum,* const int & *size*)

Sim outputToCSV function. Calls the initCSVOutput function of the sim class's stats data member. Spits processed statistics data to output.csv. Supported sizes noted below.

Precondition

function initCSVOutput must have been run before this one in order to open the fstream.

Postcondition

Processed values printed to output.csv, stored values unchanged.

Parameters

simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.

size	The size of the simulation that was run. Supported sizes for this function are 1 000, 10 000, 100 000, 1 000 000. For size 1 000 000, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.9.2.5 void simA::resetStats ( )

Sim class resetStats function. Calls the reset function of the sim class's stats data member. Resets all tracked statistics to 0.

Precondition

None.

Postcondition

Stored values reset to 0.

Returns

None.

3.9.2.6 void simA::simOne (const bool & *isNoisy,* const int & *size*)

SimA class simOne function. Runs the 1 teller 1 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

3.9.2.7 void simA::simThree (const bool & *isNoisy,* const int & *size*)

simThree function. Runs the 3 teller 1 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored in the statistics object.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

3.9.2.8 void simA::simTwo (const bool & *isNoisy,* const int & *size*)

simTwo function. Runs the 3 teller 3 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored in the statistics object.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

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

- simA.h
- simA.cpp

3.10 simN Class Reference

`#include <simN.h>`

Public Member Functions

- void simOne (const bool &, const int &)
- void simTwo (const bool &, const int &)
- void simThree (const bool &, const int &)
- void outputToConsole (const int &, const int &) const
- void initCSVOutput ()
- void outputToCSV (const int &, const int &)
- void endCSVOutput ()
- void resetStats ()

3.10.1 Detailed Description

Sim class using node based implementation for queue and priority queue data structures.. Conducts bank queue simulation operations as specified in the prompt. Uses simulation object data member to track statistics of the operation and conduct console and file output. User can call three different simulations of specified size, with full step by step of each event printed to console or summary statistic data. Pretty nifty. Supported sizes are 10, 1 000, 10 000, 100 000, 1 000 000.

3.10.2 Member Function Documentation

3.10.2.1 void simN::endCSVOutput ( )

Sim class endCSVOutput function. Calls the endCSVOutput function of the sim class's stats data member. Closes output.csv file, ending file output operations.

Precondition

None.

Postcondition

Fstream to output.csv closed.

Returns

None.

3.10.2.2 void simN::initCSVOutput ( )

Sim initCSVOutput function. Calls the initCSVOutput function of the sim class's stats data member. Opens output file (output.csv) for subsequent functions and prints a header row to the file.

Precondition

None.

Postcondition

Fstream to output.csv opened, header row printed in output.csv

Returns

None.

3.10.2.3 void simN::outputToConsole (const int & *simNum,* const int & *size*) const

Sim outputToConsole function. Calls the outputToConsole function of the sim class's stats data member. Spits processed statistics data to console. Supported sizes noted below.

Precondition

None.

Postcondition

Processed values printed to console; stored values unchanged.

Parameters

simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.
size	The size of the simulation that was run. Supported sizes for this function are 10, 1 000, 10 000, 100 000, 1 000 000. For size 10 and 1 000 000, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.10.2.4 void simN::outputToCSV (const int & *simNum,* const int & *size*)

Sim outputToCSV function. Calls the initCSVOutput function of the sim class's stats data member. Spits processed statistics data to output.csv. Supported sizes noted below.

Precondition

function initCSVOutput must have been run before this one in order to open the fstream.

Postcondition

Processed values printed to output.csv, stored values unchanged.

Parameters

simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.
size	The size of the simulation that was run. Supported sizes for this function are 1 000, 10 000, 100 000, 1 000 000. For size 1 000 000, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.10.2.5 void simN::resetStats ( )

Sim class resetStats function. Calls the reset function of the sim class's stats data member. Resets all tracked statistics to 0.

Precondition

None.

Postcondition

Stored values reset to 0.

Returns

None.

3.10.2.6 void simN::simOne (const bool & *isNoisy*, const int & *size*)

simN class simOne function. Runs the 1 teller 1 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

3.10.2.7 void simN::simThree (const bool & *isNoisy*, const int & *size*)

simThree function. Runs the 3 teller 1 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored in the statistics object.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

3.10.2.8 void simN::simTwo (const bool & *isNoisy,* const int & *size*)

simTwo function. Runs the 3 teller 3 queue simulation. Simulation conducted as per prompt. !!! ADAPTED FROM TEXTBOOK CODE !!!

Precondition

Specified correctly formatted and named data files extant.

Postcondition

Simulation has been run. Statistics data is stored in the statistics object.

Parameters

noisy	Bool denoting whether the user would like console output going step by step through the simulation.
size	The size of the simulation to be run, where size is the number of arrival events. Supported values are 10, 1 000, 10 000, 100 000, 1 000 000.

Returns

None.

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

- simN.h
- simN.cpp

3.11 statistics Class Reference

`#include <stats.h>`

Public Member Functions

- statistics ()
- void setSimulationTime (const double &)
- double getSimulationTime () const
- void setProcessingTime (const int &)
- int getProcessingTime () const
- void setAvgWaitTime (const double &)
- double getAvgWaitTime () const
- void setMaxWaitTime (const int &)
- int getMaxWaitTime () const

- void setAvgLineLength (const double &)
- double getAvgLineLength () const
- void setMaxLineLength (const int &)
- int getMaxLineLength () const
- void setIdleTime (const int &, const int &)
- int getIdleTime (const int &) const
- void reset ()
- void outputToConsole (const int &, const int &) const
- void initCSVOutput ()
- void outputToCSV (const char &, const int &, const int &)
- void endCSVOutput ()

3.11.1 Detailed Description

Statistics class. Manages and stores various operations related to tracking specified statistics on the simulation operations. Also manages output operations to console and .csv for the statistics.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 statistics::statistics ( )

Statistics class constructor. Initializes statistics class values to 0.

Precondition

None.

Postcondition

Statistics object created with zeroed out initial values.

Returns

None.

3.11.3 Member Function Documentation

3.11.3.1 void statistics::endCSVOutput ( )

Statistics class endCSVOutput function. Closes output.csv file, ending file output operations.

Precondition

None.

Postcondition

Fstream to output.csv closed.

Returns

None.

3.11.3.2 double statistics::getAvgLineLength ( ) const

Statistics class get for avgLineLength Returns value stored in avgLineLength data member.

Precondition

None.

Postcondition

Value returned.

Returns

double The value.

3.11.3.3 double statistics::getAvgWaitTime ( ) const

Statistics class get for avgWaitTime. Returns value stored in avgWaitTime data member.

Precondition

None.

Postcondition

Value returned.

Returns

double The value.

3.11.3.4 int statistics::getIdleTime (const int & *teller*) const

Statistics class get for idleTime values. Returns value stored in specified teller's idleTime data member.

Precondition

None.

Postcondition

Value for specified teller returned.

Parameters

teller	The specified teller (1 2 or 3)

Returns

int The value.

3.11.3.5 int statistics::getMaxLineLength ( ) const

Statistics class get for maxLineLength. Returns value stored in maxLineLength data member.

Precondition

None.

Postcondition

Value returned.

Returns

int The value.

3.11.3.6 int statistics::getMaxWaitTime ( ) const

Statistics class get for maxWaitTime. Returns value stored in maxWaitTime data member.

Precondition

None.

Postcondition

Value returned.

Returns

int The value.

3.11.3.7 int statistics::getProcessingTime ( ) const

Statistics class get for processingTime. Returns value stored in processingTime data member.

Precondition

None.

Postcondition

Value returned.

Returns

int The value.

3.11.3.8 double statistics::getSimulationTime ( ) const

Statistics class get for simulationTime. Returns value stored in simulationTime data member.

Precondition

None.

Postcondition

Value returned.

Returns

double The value.

3.11.3.9 void statistics::initCSVOutput ( )

Statistics class initCSVOutput function. Opens output file (output.csv) for subsequent functions and prints a header row to the file.

Precondition

None.

Postcondition

Fstream to output.csv opened, header row printed in output.csv

Parameters

dataStructure-Type	The type of data structure used in the simulation, either 'a' to signify array based or 'n' to signify node based.

Returns

None.

3.11.3.10 void statistics::outputToConsole (const int & *simNum,* const int & *size*) const

Statistics class outputToConsole function. Spits processed statistics data to console. Supported sizes noted below.

Precondition

None.

Postcondition

Processed values printed to console; stored values unchanged.

Parameters

simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.

size	The size of the simulation that was run. Supported sizes for this function are 10, 1 000, 10 000, 100 000, 1 000 000. For size 10, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.11.3.11 void statistics::outputToCSV (const char & *simType,* const int & *simNum,* const int & *size*)

Statistics class outputToCSV function. Spits processed statistics data to output.csv. Supported sizes noted below.

Precondition

function initCSVOutput must have been run before this one in order to open the fstream.

Postcondition

Processed values printed to output.csv, stored values unchanged.

Parameters

simType	The type of simulation, either 'a' to signify array based or 'n' to signify node based.
simNum	The type of simulation that was run. 1 denotes 1 teller 1 queue. 2 denotes 3 tellers 3 queues. 3 denotes 3 tellers 1 queue.
size	The size of the simulation that was run. Supported sizes for this function are 1 000, 10 000, 100 000, 1 000 000. For size 1 000 000, data is printed assuming one iteration. For the other sizes, data is printed assuming 10 iterations.

Returns

None.

3.11.3.12 void statistics::reset ( )

Statistics class reset function. Resets all tracked statistics to 0.

Precondition

None.

Postcondition

Stored values reset to 0.

Returns

None.

3.11.3.13 void statistics::setAvgLineLength (const double & *val*)

Statistics class set for avgLineLength. Sets avgLineLength data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

3.11.3.14 void statistics::setAvgWaitTime (const double & *val*)

Statistics class set for avgWaitTime. Sets avgWaitTime data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

3.11.3.15 void statistics::setIdleTime (const int & *teller,* const int & *val*)

Statistics class set for idleTime values. Sets specified teller's idle time to specified value.

Precondition

None.

Postcondition

Specified value stored in specified variable.

Parameters

teller	The specified teller (1 2 or 3)
val	The specified value.

Returns

None.

3.11.3.16 void statistics::setMaxLineLength (const int & *val*)

Statistics class set for maxLineLength. Sets maxLineLength data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

3.11.3.17 void statistics::setMaxWaitTime (const int & *val*)

Statistics class set for maxWaitTime. Sets maxWaitTime data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

3.11.3.18 void statistics::setProcessingTime (const int & *val*)

Statistics class set for processingTime. Sets processingTime data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

3.11.3.19 void statistics::setSimulationTime (const double & *val*)

Statistics class set for simulationTime. Sets simulationTime data member to specified value.

Precondition

None.

Postcondition

Specified value stored.

Parameters

val	The specified value.

Returns

None.

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

- stats.h
- stats.cpp

Chapter 4

File Documentation

4.1 event.cpp File Reference

```
#include "event.h"
```

4.1.1 Detailed Description

CS 302 Project 5 - event class implementation

Author

Patrick Austin

Date

3/21/2015

4.2 event.h File Reference

Classes

- class event

4.2.1 Detailed Description

CS 302 Project 5 - event class header

Author

Patrick Austin

Date

3/21/2015

4.3 main.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <ctime>
#include "queues.h"
#include "event.h"
#include "simA.h"
#include "simN.h"
```

Functions

- int main ()

4.3.1 Detailed Description

CS 302 Project 5 - main program for bank queue simulation

Author

Patrick Austin

Date

3/21/2015

4.3.2 Function Documentation

4.3.2.1 int main ( )

4.4 node.cpp File Reference

```
#include "node.h"
#include "event.h"
#include <cstddef>
#include <cstdlib>
```

4.4.1 Detailed Description

CS 302 Project 5 - node class implementation

Author

Patrick Austin

Date

3/21/2015

4.5 node.h File Reference

Classes

- class node< ItemType >

4.5.1 Detailed Description

CS 302 Project 5 - node class header

Author

Patrick Austin

Date

3/21/2015

4.6 priorityQueue.cpp File Reference

```
#include "queues.h"
#include "event.h"
```

4.6.1 Detailed Description

CS 302 Project 5 - sorted list and priority queue implementations

Author

Patrick Austin

Date

3/21/2015

4.7 queue.cpp File Reference

```
#include "queues.h"
#include "event.h"
#include "node.h"
```

4.7.1 Detailed Description

CS 302 Project 5 - queue implementations

Author

Patrick Austin

Date

3/21/2015

4.8 queues.h File Reference

```
#include "node.h"
```

Classes

- class arrayQueue< ItemType >
- class linkedQueue< ItemType >
- class arraySortedList< ItemType >
- class linkedSortedList< ItemType >
- class priorityQueueArray< ItemType >
- class priorityQueueLinked< ItemType >

4.8.1 Detailed Description

CS 302 Project 5 - queue, sorted list, and priority queue specifications

Author

Patrick Austin

Date

3/21/2015

4.9 randomize.cpp File Reference

```
#include "event.h"
#include "queues.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
```

Functions

- int main ()

4.9.1 Function Documentation

4.9.1.1 int main ( )

4.10 simA.cpp File Reference

```
#include "simA.h"
#include "event.h"
#include "stats.h"
#include "queues.h"
```

4.10.1 Detailed Description

CS 302 Project 5 - array based simulation class implementation

Author

Patrick Austin

Date

3/21/2015

4.11 simA.h File Reference

```
#include "stats.h"
#include "event.h"
#include "queues.h"
#include <iostream>
#include <ctime>
```

Classes

- class simA

4.11.1 Detailed Description

CS 302 Project 5 - array based simulation class specification

Author

Patrick Austin

Date

3/21/2015

4.12 simN.cpp File Reference

```
#include "simN.h"
#include "event.h"
#include "stats.h"
#include "queues.h"
```

4.12.1 Detailed Description

CS 302 Project 5 - node based simulation class implementation

Author

Patrick Austin

Date

3/21/2015

4.13 simN.h File Reference

```
#include "stats.h"
#include "event.h"
#include "queues.h"
#include <iostream>
#include <ctime>
```

Classes

- class simN

4.14 stats.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "stats.h"
```

4.14.1 Detailed Description

CS 302 Project 5 - statistics class implementation

Author

Patrick Austin

Date

3/21/2015

4.15 stats.h File Reference

```
#include <fstream>
```

Classes

- class statistics

4.15.1 Detailed Description

CS 302 Project 5 - statistics class specification

Author

Patrick Austin

Date

3/21/2015

4.16 test.cpp File Reference

```
#include "queues.h"
#include "event.h"
#include <iostream>
#include <fstream>
```

Functions

- int main ()

4.16.1 Function Documentation

4.16.1.1 int main ( )

Index