PA03 - PriorityQueue

Generated by Doxygen 1.8.6

Mon Feb 8 2016 20:15:09

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

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

4 Class Documentation

4.1 DataNode< DataType > Class Template Reference

Public Member Functions

- DataNode (const DataType &inData, DataNode< DataType > *inPrevPtr, DataNode< DataType > *inNextPtr)

  Default node constructor.

Public Attributes

- DataType **dataItem**
- DataNode< DataType > * **previous**
- DataNode< DataType > * **next**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 template<class DataType > DataNode< DataType >::DataNode (const DataType & *inData*, DataNode< DataType > * *inPrevPtr* = NULL, DataNode< DataType > * *inNextPtr* = NULL)

Default node constructor.

Constructs node with given data

Precondition

assumes DataType has default constructor & assignment operator

Postcondition

member values dataItem, previous, and next are initialized

Algorithm

initialization constructor operation

Exceptions

None	

Parameters

in	*inData*	DataType data passed into constructor

[in] inPrevPtr previous pointer for node, defaults to NULL

[in] inNextPtr next pointer for node, defaults to NULL

Returns

None

Note

None

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

- SimpleVector.h
- SimpleVector.cpp

4.2 PriorityQueue< DataType > Class Template Reference

Public Member Functions

- PriorityQueue (int initialCapacity=DEFAULT_CAPACITY, int numPriorities=DEFAULT_NUM_PRIORITIES)
  Default/Initialization constructor.
- PriorityQueue (const PriorityQueue &copiedQueue)
  Copy constructor.
- ~PriorityQueue ()
  Class destructor.
- const PriorityQueue & operator= (const PriorityQueue< DataType > &rhPQueue)
  Object assignment operator.
- void enqueue (const DataType &dataItem) throw (logic_error)
  Enqueue operation.
- void dequeue (DataType &dQData) throw (logic_error)
  Dequeue operation.
- void peekAtFront (DataType &pkData) throw (logic_error)
  Peek at front operation.
- bool isEmpty () const
  Checks for empty list operation.
- void showStructure (char ID)
  Displays queue as presently implemented.

Static Public Attributes

- static const int **DEFAULT_CAPACITY** = 10
- static const int **DEFAULT_NUM_PRIORITIES** = 5
- static const int **DATA_SET_STR_LEN** = 100
- static const char **TAB** = '\t'
- static const char **SPACE** = ' '

Private Attributes

- int **maxPriorities**
- UtilityVector< DataType > **qData**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 template<class DataType > PriorityQueue< DataType >::PriorityQueue (int *initialCapacity* = DEFAULT_CAPACITY, int *numPriorities* = DEFAULT_NUM_PRIORITIES)

Default/Initialization constructor.

Constructs PriorityQueue with either default or given capacity and number of priorities

Precondition

Assumes Uninitialized PriorityQueue object

Postcondition

Initializes priority and UtilityVectorData

Algorithm

Initializes class by assigning priority and initializing UtilityVector

Exceptions

None	

Parameters

in	*initialCapacity*	Desired default or user-provided capacity (int)
in	*numPriorities*	Number of priorities to be used (int)

Returns

None

Note

The incorrect parameter, numPriorities, is used in the default constructor of the UtilityVector object so as to get the output to match the Submit system

4.2.1.2 template<class DataType > PriorityQueue< DataType >::PriorityQueue (const PriorityQueue< DataType > & *copiedQueue*)

Copy constructor.

Constructs PriorityQueue as a copy of another PriorityQueue object

Precondition

Assumes uninitialized PriorityQueue object

Postcondition

Initializes priority and UtilityVector data

Algorithm

Initializes class by copying data from other PriorityQueue object

Exceptions

None	

Parameters

in	*copiedQueue*	Other PriorityQueue object (PriorityQueue<DataType>)

Returns

None

Note

None

4.2.1.3 template<class DataType > PriorityQueue< DataType >::~PriorityQueue ( )

Class destructor.

Destructor of data member qData called to remove data

Precondition

Assumes initialized PriorityQueue object

Postcondition

Local data is removed and UtilityVector object is destructed

Algorithm

UtilityVector object is destructed implicitly upon destruction of this object

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.2.2 Member Function Documentation

4.2.2.1 template<class DataType > void PriorityQueue< DataType >::dequeue (DataType & *dQData*) throw logic_error)

Dequeue operation.

If list is not empty, removes item at front of queue (which is at first element of the vector) and passes it back

Precondition

Assumes initialized PriorityQueue object

Postcondition

Data is dequeued from head, data is passed to calling function

Algorithm

Removes data from index zero of UtilityVector using removeAtIndex

Exceptions

logic_error	Throws exception if empty list

Parameters

out	*dQData*	Data item that has been dequeued (DataType)

Returns

None

Note

None

4.2.2.2 template<class DataType > void PriorityQueue< DataType >::enqueue (const DataType & *dataItem*) throw logic_error)

Enqueue operation.

Enqueues data item at appropriate priority in PriorityQueue

Precondition

Assumes initialized PriorityQueue object

Postcondition

Data is enqueued at appropriate priority level

Algorithm

Finds priority level to insert item using DataType method getPriority, then inserts item using UtilityVector method insertAtIndex

Exceptions

logic_error	Throws exception if incorrect priority value

Parameters

in	*dataItem*	New item to be enqueued (DataType)

Returns

None

Note

None

4.2.2.3 template<class DataType > bool PriorityQueue< DataType >::isEmpty ( ) const

Checks for empty list operation.

Checks with UtilityVector for empty list

Precondition

Assumes initialized PriorityQueue object

Postcondition

Returns evidence of empty list

Algorithm

Tests UtilityVector for zero list size

Exceptions

None	

Parameters

None	

Returns

Boolean evidence of empty PriorityQueue

Note

None

4.2.2.4 template<class DataType > const PriorityQueue< DataType > & PriorityQueue< DataType >::operator= (const PriorityQueue< DataType > & *rhPQueue*)

Object assignment operator.

Copies other PriorityQueue object into this local object using overloaded assignment operator

Precondition

Assumes initialized PriorityQueue object

Postcondition

Local PriorityQueue object holds copy of assigned object

Algorithm

If right-hand object is not the same as the local this object, copies number of maxPriorities and uses UtilityVector assignment operation to transfer data

Exceptions

None	

Parameters

in	*rhPQueue*	Other object to be assigned (PriorityQueue<DataType>)

Returns

Reference to this object

Note

None

4.2.2.5 template<class DataType > void PriorityQueue< DataType >::peekAtFront (DataType & *pkData*) throw logic_error)

Peek at front operation.

If list is not empty, acquires data at front of queue (which is at first element of the vector) and passes it back

Precondition

Assumes initialized PriorityQueue object

Postcondition

Data found at head is passed to calling function

Algorithm

Returns the data from index zero of UtilityVector by accessing the first element in the UtilityVector

Exceptions

logic_error	Throws exception if empty list

Parameters

out	*pkData*	Reference parameter that holds data item that has been acquired at the first element in the UtilityVector (DataType)

Returns

None

Note

None

4.2.2.6 template<class DataType > void PriorityQueue< DataType >::showStructure (char *listID*)

Displays queue as presently implemented.

Allows for character parameter to identify list to user

Precondition

Assumes initialized PriorityQueue object

Postcondition

Displays either empty list indication or PriorityQueue data

Algorithm

Iterates through PriorityQueue from beginning to end

Exceptions

None	

Parameters

in	*listID*	Identifies which object data is being displayed (char)

Returns

None

Note

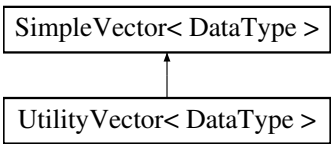
None

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

- PriorityQueue.h
- PriorityQueue.cpp

4.3 SimpleVector< DataType > Class Template Reference

Inheritance diagram for SimpleVector< DataType >:



Public Member Functions

- SimpleVector (int newCapacity=DEFAULT_CAPACITY)
  Default/Initialization SimpleVector constructor.
- SimpleVector (int newCapacity, const DataType &fillValue)
  Initialization fill constructor.
- SimpleVector (const SimpleVector< DataType > &copiedVector)
  Copy constructor.
- ~SimpleVector ()
  object destructor
- const SimpleVector< DataType > & operator= (const SimpleVector< DataType > &rhVector)
  Overloaded assignment operation.
- int getCapacity () const
  SimpleVector capacity accessor.
- int getSize () const
  SimpleVector size accessor.

- void showSVStructure (char IDChar)

  Shows structure of list as array.
- void setAtIndex (int index, const DataType &inData) throw (logic_error)

  SimpleVector set element data method.
- const DataType & getAtIndex (int index) throw (logic_error)

  SimpleVector get element data method.
- void resize (int newCapacity)

  SimpleVector resize (i.e., change capacity) operation.
- void incrementSize ()

  SimpleVector size mutator - increase.
- void decrementSize ()

  SimpleVector size mutator - decrease.
- void zeroSize ()

  SimpleVector size mutator - zero.

Static Public Attributes

- static const int **LARGE_STR_LEN** = 100
- static const int **DEFAULT_CAPACITY** = 10
- static const int **DISPLAY_WIDTH** = 5
- static const char **SPACE** = ' '
- static const char **COLON** = ':'
- static const char **LEFT_BRACKET** = '['
- static const char **RIGHT_BRACKET** = ']'

Private Member Functions

- void copyVectorObject (const SimpleVector< DataType > &inData)

  SimpleVector copy utility.
- DataNode< DataType > * getPointerToIndex (int index)

  SimpleVector array element access utility.

Private Attributes

- int **vectorCapacity**
- int **vectorSize**
- int **currentIndex**
- DataNode< DataType > * **currentPtr**
- DataNode< DataType > * **listHead**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 template<class DataType > SimpleVector< DataType >::SimpleVector (int *newCapacity* = DEFAULT_CAPACITY)

Default/Initialization SimpleVector constructor.

Constructs SimpleVector with either default or given capacity

Precondition

assumes uninitialized SimpleVector object

Postcondition

list of nodes is created for use as array
member values vectorCapacity and vectorSize are first initialized in the constructor
member values vectorCapacity, vectorSize, currentIndex, currentPtr, and listHead are initialized in resize

Algorithm

sets initial values to start resize, then calls resize

Exceptions

None	

Parameters

in	*newCapacity*	desired default or user-provided capacity

Returns

None

Note

None

4.3.1.2 template<class DataType > SimpleVector< DataType >::SimpleVector (int *newCapacity,* const DataType & *fillValue*)

Initialization fill constructor.

Constructs object with all elements filled

Precondition

assumes uninitialized SimpleVector object

Postcondition

list of nodes is created for use as array
member values vectorCapacity and vectorSize are first initialized in the constructor
member values vectorCapacity, vectorSize, currentIndex, currentPtr, and listHead are initialized in resize

Algorithm

sets initial values to start resize, then calls resize, then fills all nodes with data, sets vectorSize to vectorCapacity

Exceptions

None	

Parameters

in	*newCapacity*	user-defined object capacity

Returns

None

Note

None

4.3.1.3 template<class DataType > SimpleVector< DataType >::SimpleVector (const SimpleVector< DataType > & *copiedVector*)

Copy constructor.

Creates local copy of all contents of parameter object

Precondition

Assumes uninitialized SimpleVector object

Postcondition

member values vectorCapacity and vectorSize are first initialized in the constructor
member values vectorCapacity, vectorSize, currentIndex, currentPtr, and listHead are set in copyVectorObject

Algorithm

sets initial values to start copyVectorObject, then calls copyVectorObject, which sets vectorCapacity, vectorSize, currentIndex, currentPtr

Exceptions

None	

Parameters

in	*copiedVector*	incoming Vector object

Returns

None

Note

None

4.3.1.4 template<class DataType > SimpleVector< DataType >::~SimpleVector ( )

object destructor

removes or verifies removal of all data in SimpleVector

Precondition

assumes SimpleVector capacity >= 0

Postcondition

all linked list nodes are removed, using resize

Algorithm

calls resize function, which handles all conditions

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.2 Member Function Documentation

4.3.2.1 template<class DataType > void SimpleVector< DataType >::copyVectorObject (const SimpleVector< DataType > & *inData*) [private]

SimpleVector copy utility.

Copies the data from a complete object into this object

Precondition

No assumption of initialization

Postcondition

Object contains copy of data and states from copied object

Algorithm

this object is resized to copied object capacity if copied object's capacity > 0, copies head data, then copies subsequent elements as needed, updates current index and pointer during copy copies copied object size to this object, copies copied object index and related pointer to this object

Exceptions

None	

Parameters

in	*copied*	SimpleVector object

Returns

None

Note

Overwrites any data previously in this object

4.3.2.2 template<class DataType > void SimpleVector< DataType >::decrementSize ( )

SimpleVector size mutator - decrease.

decreases SimpleVector size count; has no impact on data

Precondition

Assumes SimpleVector initialize to capacity >= 0

Postcondition

SimpleVector size value is decremented

Algorithm

Decrement size value

Exceptions

None	

Parameters

None	

Returns

None

Note

Provided as convenience for user; has no impact on SimpleVector data

4.3.2.3 template<class DataType > const DataType & SimpleVector< DataType >::getAtIndex (int *index*) throw logic_error)

SimpleVector get element data method.

allows assignment of data to element in this SimpleVector

Precondition

Assumes initialized SimpleVector

Postcondition

Returns value at index as const quantity

Algorithm

Finds node related to index, returns value

Exceptions

throws	logic error if index is out of bounds

Parameters

in	*index*	of element to be retrieved

Returns

Copy of data value at index

Note

None

4.3.2.4 template<class DataType > int SimpleVector< DataType >::getCapacity ( ) const

SimpleVector capacity accessor.

None

Precondition

SimpleVector has some capacity >= 0

Postcondition

No change in data, capacity returned

Algorithm

returns vectorCapacity as value

Exceptions

None	

Parameters

None	

Returns

SimpleVector capacity

Note

None

4.3.2.5 template<class DataType > DataNode< DataType > * SimpleVector< DataType >::getPointerToIndex (int *index*) [private]

SimpleVector array element access utility.

Specified element data accessed by index and returned

Precondition

Assumes initialized SimpleVector where 0 <= index < vectorCapacity

Postcondition

Returns object at index

Algorithm

Identifies requested index position closest to current index position, moves index and node pointer to that position

Algorithm

If new index > current index and distance to new index < vectorCapacity /2, increments upward

Algorithm

If new index < current index and distance to new index > vectorCapacity /2, increments upward

Algorithm

If new index < current index and distance to new index < vectorCapacity /2, increments downward

Algorithm

If new index > current index and distance to new index > vectorCapacity /2, increments upward

Exceptions

None	

Parameters

in	*index*	index of element to be accessed

Returns

pointer to data item, or NULL, as specified

Note

None

4.3.2.6 template<class DataType > int SimpleVector< DataType >::getSize ( ) const

SimpleVector size accessor.

None

Precondition

SimpleVector has some size >= 0

Postcondition

No change in data, size returned

Algorithm

returns vectorSize as value

Exceptions

None	

Parameters

None	

Returns

SimpleVector size

Note

None

4.3.2.7 template<class DataType > void SimpleVector< DataType >::incrementSize ( )

SimpleVector size mutator - increase.

increases SimpleVector size count; has no impact on data

Precondition

Assumes SimpleVector initialize to capacity >= 0

Postcondition

SimpleVector size value is incremented

Algorithm

Increment size value

Exceptions

None	

Parameters

None	

Returns

None

Note

Provided as convenience for user; has no impact on SimpleVector data

4.3.2.8 template<class DataType > const SimpleVector< DataType > & SimpleVector< DataType >::operator= (const SimpleVector< DataType > & *rhVector*)

Overloaded assignment operation.

Assigns data from right-hand object to this object

Precondition

no assumptions made about this object prior to assignment

Postcondition

object contains a complete data copy of assigned right-hand object

Algorithm

checks for not assigning to self, then calls copyVectorObject, which handles all condtions

Exceptions

None	

Parameters

in	*rhVector*	SimpleVector object to be assigned

Returns

Reference to this object

Note

None

4.3.2.9 template<class DataType > void SimpleVector< DataType >::resize (int *newCapacity*)

SimpleVector resize (i.e., change capacity) operation.

Changes SimpleVector capacity to amount given in parameter

Precondition

Assumes SimpleVector initialized to capacity >= 0

Postcondition

SimpleVector capacity is changed to requested amount

Algorithm

For condition: empty SimpleVector and newCapacity > 0, starts by creating head node

Algorithm

For condition: newCapacity > vectorCapacity, adds nodes as needed, updates vectorCapacity

Algorithm

For condition: newCapacity < vectorCapacity and vectorCapacity > 1, removes nodes previous to head, updates vectorCapacity

Algorithm

For condition: newCapacity == 0, removes last node, sets head to NULL, vectorCapacity to 0

Algorithm

For all conditions: resets index to zero and related node pointer to head

Algorithm

For condition: empty SimpleVector and newCapacity == 0, does nothing

Exceptions

None	

Parameters

in	*new*	capacity requested

Returns

None

Note

Makes no distinction about stored data; if capacity is reduced, data may be lost

4.3.2.10 template<class DataType > void SimpleVector< DataType >::setAtIndex (int *index*, const DataType & *inData*) throw logic_error)

SimpleVector set element data method.

allows assignment of data to element in this SimpleVector

Precondition

Assumes initialized SimpleVector

Postcondition

Assigns new value to element and/or returns value

Algorithm

Finds node related to index, assigns data to node

Exceptions

throws	logic error if index is out of bounds

Parameters

in	*index*	index of element to be assigned
in	*inData*	new data to be set at index

Returns

None

Note

None

4.3.2.11 template<class DataType > void SimpleVector< DataType >::showSVStructure (char *IDChar*)

Shows structure of list as array.

None

Precondition

Assumes initialized SimpleVector where 0 <= index < vectorCapacity

Postcondition

Provides display as specified

Algorithm

Iterates across linked list, showing data items as elements

Exceptions

None	

Parameters

in	*IDChar*	character ID letter to indicate object displayed

Returns

None

Note

None

4.3.2.12 template<class DataType > void SimpleVector< DataType >::zeroSize ( )

SimpleVector size mutator - zero.

Sets SimpleVector size count to zero; has no impact on data

Precondition

Assumes SimpleVector initialize to capacity >= 0

Postcondition

SimpleVector size value is set to zero

Algorithm

Set size value to zero

Exceptions

None	

Parameters

None	

Returns

None

Note

Provided as convenience for user; has no impact on SimpleVector data

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

- SimpleVector.h
- SimpleVector.cpp

4.4 StudentType Class Reference

Public Member Functions

- StudentType ()

Default/Initialization constructor.

- StudentType (char *studentName, int univIDNum, char *univClassLevel)

  Initialization constructor.
- const StudentType & operator= (const StudentType &rhStudent)

  Assignment operation.
- void setStudentData (char *studentName, int studentID, char *studentLevel)

  Data setting utility.
- int compareTo (const StudentType &otherStudent) const

  Data comparison utility.
- void toString (char *outString) const

  Data serialization.
- int getPriority () const

  Gets numerical priority related to priority letter (char)

Static Public Attributes

- static const int **STD_STR_LEN** = 50
- static const int **DATA_SET_STR_LEN** = 100
- static const char **NULL_CHAR** = '\0'

Private Member Functions

- int setPriority (char *priorityString)

  Sets numerical priority related to priority letter (char)
- void copyString (char *destination, const char *source)

  String copy utility.

Private Attributes

- char **name** [STD_STR_LEN]
- int **universityID**
- int **priority**

4.4.1 Constructor & Destructor Documentation

4.4.1.1 StudentType::StudentType ( )

Default/Initialization constructor.

Constructs StudentType with default data

Precondition

assumes uninitialized StudentType object

Postcondition

Initializes all data quantities

Algorithm

Initializes class by assigning name, Id number, and class level

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.4.1.2 StudentType::StudentType (char * *studentName,* int *univIDNum,* char * *univClassLevel*)

Initialization constructor.

Constructs StudentType with provided data

Precondition

assumes uninitialized StudentType object, assumes string max length < STD_STR_LEN

Postcondition

Initializes all data quantities

Algorithm

Initializes class by assigning name, Id number, and class level

Exceptions

None	

Parameters

in	*studentName*	Name of student as c-string
in	*univIDNum*	University ID number as integer
in	*univClassLevel*	University class/grade level

Returns

None

Note

None

4.4.2 Member Function Documentation

4.4.2.1 int StudentType::compareTo (const StudentType & *otherStudent*) const

Data comparison utility.

Provides public comparison operation for use in other classes

Precondition

Makes no assumption about StudentType data

Postcondition

Provides integer result of comparison such that:

- result $<$ 0 indicates this $<$ other
- result == 0 indicates this == other
- result $>$ 0 indicates this $>$ other

Algorithm

Sets priorities of this and other class level item, then provides mathematic difference

Exceptions

None	

Parameters

in	*otherStudent*	Other student data to be compared to this object

Returns

Integer result of comparison process

Note

None

4.4.2.2 void StudentType::copyString (char * *destination*, const char * *source*) [private]

String copy utility.

Copies source string into destination string

Precondition

assumes standard string conditions, including NULL_CHAR end

Postcondition

desination string holds copy of source string

Algorithm

Copies string character by character until end of string character is found, assumes string max length $<$ STD-_STR_LEN

Exceptions

None	

Parameters

out	*Destination*	string
in	*Source*	string

Returns

None

Note

None

4.4.2.3 int StudentType::getPriority ( ) const

Gets numerical priority related to priority letter (char)

None

Precondition

makes no assumptions about priority data

Postcondition

provides priority value related to letter/char parameter

Algorithm

Uses lookup table to set priorities

Exceptions

None	

Parameters

in	*student*	level in string form

Returns

Integer result of priority letter lookup

Note

None

4.4.2.4 const StudentType & StudentType::operator= (const StudentType & *rhStudent*)

Assignment operation.

Class overloaded assignment operator

Precondition

assumes initialized other object

Postcondition

desination object holds copy of local this object

Algorithm

Copies each data item separately

Exceptions

None	

Parameters

in	*rhStudent*	other StudentType object to be assigned

Returns

Reference to local this StudentType object

Note

None

4.4.2.5 int StudentType::setPriority (char * *priorityString*) [private]

Sets numerical priority related to priority letter (char)

None

Precondition

makes no assumptions about priority data

Postcondition

provides priority value related to letter/char parameter

Algorithm

Uses lookup table to set priorities

Exceptions

None	

Parameters

in	*student*	level in string form

Returns

Integer result of priority letter lookup

Note

None

4.4.2.6 void StudentType::setStudentData (char * *studentName,* int *studentID,* char * *classLevel*)

Data setting utility.

Allows resetting data in StudentType

Precondition

Makes no assumption about StudentType data

Postcondition

Data values are correctly assigned in StudentType

Algorithm

Assigns data values to class members

Exceptions

None	

Parameters

in	*studentName*	String name of student
in	*studentID*	Integer value of student ID
in	*studentLevel*	String name of student

Returns

Integer result of comparison process

Note

None

4.4.2.7 void StudentType::toString (char * *outString*) const

Data serialization.

Converts data set to string for output by other data types

Precondition

Assumes data is initialized

Postcondition

Provides all data as string

Algorithm

Places data into formatted string

Exceptions

None	

Parameters

out	*outString*	string containing class data

Returns

None

Note

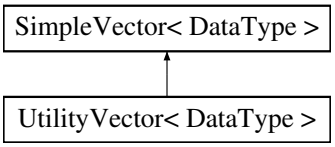
None

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

- StudentType.h
- StudentType.cpp

4.5 UtilityVector< DataType > Class Template Reference

Inheritance diagram for UtilityVector< DataType >:



Public Member Functions

- UtilityVector (int newCapacity=DEFAULT_CAPACITY)
  Default/Initialization UtilityVector constructor.
- UtilityVector (int newCapacity, const DataType &fillValue)
  Initialization fill constructor.
- UtilityVector (const UtilityVector< DataType > &copiedVector)
  Copy constructor.
- ~UtilityVector ()
  Object destructor.
- void copyFromTo (int indexTo, int indexFrom)
  Copies data between elements.
- void swapBetween (int oneIndex, int otherIndex)
  Swaps data between elements.
- void insertAtIndex (int insertIndex, const DataType &itemToInsert)
  Inserts DataItem at specified index.
- void removeAtIndex (int removalIndex, DataType &removedItem)
  Removes DataItem at specified index.

Static Public Attributes

- static const int **DEFAULT_CAPACITY** = 10

4.5.1 Constructor & Destructor Documentation

4.5.1.1 template<typename DataType > UtilityVector< DataType >::UtilityVector (int *newCapacity* = DEFAULT_CAPACITY)

Default/Initialization UtilityVector constructor.

Constructs UtilityVector with either default or given capacity

Precondition

Assumes Uninitialized UtilityVector object

Postcondition

List of nodes is created for use as array

Algorithm

Initializes class by calling parent constructor

Exceptions

None	

Parameters

in	*newCapacity*	Desired default or user-provided capacity (int)

Returns

None

Note

None

4.5.1.2 template<class DataType > UtilityVector< DataType >::UtilityVector (int *newCapacity*, const DataType & *fillValue*)

Initialization fill constructor.

Constructs object with all elements filled

Precondition

Assumes uninitialized UtilityVector object

Postcondition

list of nodes is created for use as array

Algorithm

Initializes class by calling parent constructor

Exceptions

None	

Parameters

in	*newCapacity*	User-defined object capacity (int)
in	*fillValue*	DataType fill value (DataType)

Returns

None

Note

None

4.5.1.3 template<typename DataType > UtilityVector< DataType >::UtilityVector (const UtilityVector< DataType > & *copiedVector*)

Copy constructor.

Creates local copy of all contents of parameter object

Precondition

Assumes uninitialized UtilityVector object

Algorithm

Calls parent constructor for copy process

Exceptions

None	

Parameters

in	*copiedVector*	Incoming Vector object (UtilityVector<DataType>)

Returns

None

Note

None

4.5.1.4 template<typename DataType > UtilityVector< DataType >::~UtilityVector ( )

Object destructor.

Removes or verifies removal of all data in UtilityVector

Precondition

Assumes UtilityVector capacity >= 0

Algorithm

Calls parent destructor

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.5.2 Member Function Documentation

4.5.2.1 template<typename DataType > void UtilityVector< DataType >::copyFromTo (int *indexTo,* int *indexFrom*)

Copies data between elements.

Copies DataType value from one vector element to another using indices

Precondition

Assumes data found in elements, and that vectorCapacity > indexFrom and vectorCapacity > indexTo

Postcondition

vector element at indexTo contains the data found at element at indexFrom

Algorithm

Acquires data using getAtIndex, assigns data using setAtIndex

Exceptions

Boundary	Exception called if from or to indices are out of bounds

Parameters

in	*indexTo*	Index for element to which data is copied (int)
in	*indexFrom*	Index for element from which data is copied (int)

Returns

None

Note

None

4.5.2.2 template<typename DataType > void UtilityVector< DataType >::insertAtIndex (int *insertIndex,* const DataType & *itemToInsert*)

Inserts DataItem at specified index.

Shifts all data above inserted location up

Precondition

Assumes vectorSize data is correct
Assumes data found in elements, vectorCapacity > vectorSize

Postcondition

All data is moved up from insertion location, given DataType item inserted at insertion location

Algorithm

Copies data from each element up to next element using copyFromTo, inserts item using setAtIndex

Exceptions

Boundary	Exception called if one or other indices are out of bounds

Parameters

in	*insertIndex*	Index for element of element to acquire inserted data (int)
in	*itemToInsert*	Data item to be inserted into vector (int)

Returns

None

Note

None

4.5.2.3 template<typename DataType > void UtilityVector< DataType >::removeAtIndex (int *removalIndex,* DataType & *removedItem*)

Removes DataItem at specified index.

Shifts all data above removed location down

Precondition

Assumes vectorSize data is correct
Assumes data found in elements

Postcondition

All data is moved down by one element to the removal location, given DataType item removed at removal location, and passed back to calling function

Algorithm

Acquired data item from element Copies data from each element down by one element to the removed index using copyFromTo, passes item back to calling function

Exceptions

Boundary	Exception called if one or other indices are out of bounds

Parameters

in	*removalIndex*	Index of element to be removed from vector (int)
out	*removedItem*	Data removed from vector and passed back to calling function (DataType)

Returns

None

Note

None

4.5.2.4 template<typename DataType > void UtilityVector< DataType >::swapBetween (int *oneIndex,* int *otherIndex*)

Swaps data between elements.

Acquires DataType quantities from two elements, swaps between them

Precondition

Assumes data found in elements, and that vectorCapacity > oneIndex and vectorCapacity > otherIndex

Postcondition

Vector element at oneIndex contains the data found at element at otherIndex, and vector element at otherIndex contains data found at oneIndex

Algorithm

Acquires data for both items using getAtIndex, assigns data to opposite indices using setAtIndex

Exceptions

Boundary	Exception called if one or other indices are out of bounds

Parameters

in	*oneIndex*	Index for element of one of two elements to be swapped (int)
in	*otherIndex*	Index for element of other of two elements to be swapped (int)

Returns

None

Note

None

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

- UtilityVector.h
- UtilityVector.cpp

5 File Documentation

5.1 PA03.cpp File Reference

Driver program to exercise the PriorityQueue class.

```
#include <iostream>
#include <cstring>
#include "StudentType.h"
#include "SimpleVector.cpp"
#include "UtilityVector.cpp"
#include "PriorityQueue.cpp"
```

Functions

- void ShowMenu ()

  ShowMenu: Displays choice of commands for exercising priority queue.
- char GetCommandInput (char *nameString,int &studentID,char *gradeLevel)

  GetCommandInput: Acquires command input from user.
- int **main** ()

Variables

- const int **LARGE_STR_LEN** = 100
- const int **SMALL_STR_LEN** = 25
- const bool **VERBOSE** = true
- const char **SEMICOLON** = ';'
- const char **ENDLINE_CHAR** = '\n'
- const char **PERIOD** = '.'
- const int **TEST_PQ_NUM_PRIORITIES** = 12

5.1.1 Detailed Description

Driver program to exercise the PriorityQueue class. Allows for testing all PriorityQueue methods in an interactive environment

Version

1.10 Michael Leverington (30 January 2016) Updated for use with UtilityVector

1.00 Michael Leverington (07 September 2015) Original code

Requires SimpleVector.cpp, UtilityVector.cpp, StudentType.h, PriorityQueue.cpp

5.1.2 Function Documentation

5.1.2.1 char GetCommandInput (char * *nameString*, int & *studentID*, char * *gradeLevel*)

GetCommandInput: Acquires command input from user.

Command letters are unique combinations of three letters

Parameters

None	

Note

Clears input string, loads command letters individually using extraction operation; adds input character for display and output line for display clearance

5.1.2.2 void ShowMenu ( )

ShowMenu: Displays choice of commands for exercising priority queue.

Command letters displayed indicate operations to be conducted

Parameters

None	

Note

None

5.2 PriorityQueue.cpp File Reference

Implementation file for PriorityQueue class.

```
#include "UtilityVector.h"
#include "PriorityQueue.h"
```

Variables

- static const float **HALF** = 0.50
- static const float **FOURTH** = 0.25

5.2.1 Detailed Description

Implementation file for PriorityQueue class. Implements all member methods of the PriorityQueue class

Version

1.00 Bryan Kline (08 February 2016)

Requires PriorityQueue.h

5.3 PriorityQueue.h File Reference

Definition file for PriorityQueue class.

```
#include <stdexcept>
#include <iostream>
#include "StudentType.h"
#include "UtilityVector.h"
```

Classes

- class PriorityQueue< DataType >

5.3.1 Detailed Description

Definition file for PriorityQueue class. Specifies all member methods of the PriorityQueue class, which uses the UtilityVector class

Version

1.10 Michael Leverington (30 January 2016) Updated for use with UtilityVector

1.00 Michael Leverington (07 September 2015) Original code

None

5.4 SimpleVector.cpp File Reference

Implementation file for SimpleVector class.

```
#include "SimpleVector.h"
```

5.4.1 Detailed Description

Implementation file for SimpleVector class.

Author

Michael Leverington

Implements all member methods of the SimpleVector class

Version

1.10 Michael Leverington (19 January 2016) Updated for use with linked list

1.00 Michael Leverington (30 August 2015) Original code

Requires SimpleVector.h

5.5 SimpleVector.h File Reference

Definition file for SimpleVector class.

```
#include <iostream>
#include <stdexcept>
#include <cstdlib>
```

Classes

- class DataNode< DataType >
- class SimpleVector< DataType >

5.5.1 Detailed Description

Definition file for SimpleVector class. Specifies all member methods of the SimpleVector class

Version

1.10 Michael Leverington (19 January 2016) Updated for use with linked list

1.00 Michael Leverington (30 August 2015) Original code

None

5.6 StudentType.cpp File Reference

Implementation file for StudentType class.

```
#include "StudentType.h"
#include <cstdio>
#include <iostream>
```

5.6.1 Detailed Description

Implementation file for StudentType class. Implements the constructor method of the StudentType class

Version

1.00 (07 September 2015)

Requires StudentType.h

5.7 UtilityVector.cpp File Reference

Implementation file for UtilityVector.

```
#include "UtilityVector.h"
#include "SimpleVector.h"
```

5.7.1 Detailed Description

Implementation file for UtilityVector.

Author

Bryan Kline

Implements all member methods of the UtilityVector

Version

1.00 (08 February 2016)

Requires UtilityVector.h, SimpleVector.h

5.8 UtilityVector.h File Reference

Definition file for UtilityVector class.

```
#include <iostream>
#include <stdexcept>
#include <cstdlib>
#include "SimpleVector.h"
```

Classes

- class UtilityVector< DataType >

5.8.1 Detailed Description

Definition file for UtilityVector class. Specifies all member methods of the UtilityVector class

Version

1.00 Michael Leverington (29 January 2016) Original code

Requires SimpleVector.h

Index