PA05 - SorterClass

Generated by Doxygen 1.8.6

Wed Feb 24 2016 14:14:03

Contents

1 Hierarchical Index

1.1 Class Hierarchy

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

- **DataNode< DataType >** 7
- **DataNode< StudentType >** 7
- **SimpleTimer** 19
- **SimpleVector< DataType >** 20
  - **SorterClass< DataType >** 32
- **SimpleVector< StudentType >** 20
  - **SorterClass< StudentType >** 32
    - **BblSorter** 3
    - **MrgSorter** 8

2 Class Index

2.1 Class List

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

3 File Index

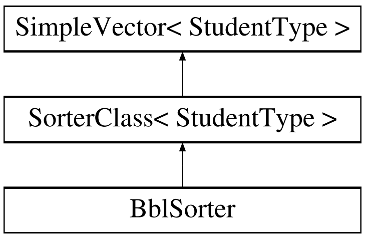
3.1 File List

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

4 Class Documentation

4.1 BblSorter Class Reference

Inheritance diagram for BblSorter:



Public Member Functions

- BblSorter ()

  Implementation of BblSorter default constructor.
- BblSorter (int initialCapacity)

  Implementation of BblSorter parameterized constructor.
- BblSorter (const SorterClass< StudentType > &copiedSorter)

  Implementation of BblSorter copy constructor.
- virtual ∼BblSorter ()

  Implementation of BblSorter destructor.
- virtual void sort ()

  Implementation of BblSorter method to sort items in the vector.

Private Member Functions

- bool sortHelper (int index)

  Implementation of BblSorter method to assist in sorting the items in the vector.

Additional Inherited Members

4.1.1 Constructor & Destructor Documentation

4.1.1.1 BblSorter::BblSorter ( )

Implementation of BblSorter default constructor.

The base class default constructor is called with an initializer

Precondition

Assumes an uninitialized BblSorter object

Postcondition

An uninitialized BblSorter object

Algorithm

The base class default constructor is called with an initializer

Exceptions

None	

Parameters

None	

Returns

None

Note

Initializer used

4.1.1.2 BblSorter::BblSorter (int *initialCapacity*)

Implementation of BblSorter parameterized constructor.

The base class parameterized constructor is called with an initializer

Precondition

Assumes an uninitialized BblSorter object

Postcondition

An initialized BblSorter object with nodes created for the vector in the amount of the parameter passed in

Algorithm

The base class parameterized constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*initialCapacity*	An int corresponding to the number of nodes that will be created for the vector (int)

Returns

None

Note

Initializer used

4.1.1.3 BblSorter::BblSorter (const SorterClass< StudentType > & *copiedSorter*)

Implementation of BblSorter copy constructor.

The base class copy constructor is called with an initializer

Precondition

Assumes an uninitialized BblSorter object

Postcondition

A BblSorter object with the same nodes and values as the object passed in as a parameter

Algorithm

The base class copy constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*copiedSorter*	A const referenced parameter that corresponds to the BblSorter object to be copied into the local object (SorterClass<StudentType>)

Returns

None

Note

Initializer used

4.1.1.4 BblSorter::∼BblSorter ( ) `[virtual]`

Implementation of BblSorter destructor.

Destructs the BblSorter object

Precondition

Assumes an initialized BblSorter object with nodes

Postcondition

All memory allocated to nodes is freed and data members are set to default values

Algorithm

The base class method resize is called with an argument of zero to clear all nodes and set data members to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.1.2 Member Function Documentation

4.1.2.1 void BblSorter::sort ( ) `[virtual]`

Implementation of BblSorter method to sort items in the vector.

The method sortHelper is called to sort the items in ascending order

Precondition

Assumes an initialized BblSorter object with nodes which will hold objects of type StudentType

Postcondition

The vector has its items arranged in ascending order

Algorithm

A counter controlled loop calls the method sortHelper while a bool corresponding to whether or not there has been a swap is true

Exceptions

None	

Parameters

None	

Returns

None

Note

The items in the vector must have a compareTo method for the sortHelper method

Reimplemented from SorterClass< StudentType >.

4.1.2.2 bool BblSorter::sortHelper (int *index*) [private]

Implementation of BblSorter method to assist in sorting the items in the vector.

The items in the vector are sorted with recursive bubble sort into ascending order

Precondition

Assumes an initialized BblSorter object with nodes which will hold objects of type StudentType

Postcondition

The vector has its items arranged in ascending order

Algorithm

An if statement checks if the current index is at the end of the vector, which is the base case, and if not then an if statement checks whether the item at the next index is greater than the one at the current index, with a call to the StudentType method compareTo, and if so the items are swapped with a call to the base class method swapBetween, the method is then called again recursively until hitting the base case

Exceptions

None	

Parameters

None	

Returns

None

Note

The items in the vector must have a compareTo method

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

- BblSorter.h
- BblSorter.cpp

4.2 DataNode< DataType > Class Template Reference

Public Member Functions

- DataNode (const DataType &inData, DataNode< DataType > ∗inPrevPtr=NULL, DataNode< DataType > ∗inNextPtr=NULL)

  Default node constructor.

Public Attributes

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

4.2.1 Constructor & Destructor Documentation

4.2.1.1 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

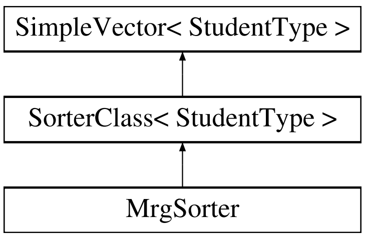
None

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

- SimpleVector.h
- SimpleVector.cpp

4.3 MrgSorter Class Reference

Inheritance diagram for MrgSorter:



Public Member Functions

- MrgSorter ()

  Implementation of MrgSorter default constructor.
- MrgSorter (int initialCapacity)

  Implementation of MrgSorter parameterized constructor.
- MrgSorter (const SorterClass< StudentType > &copiedSorter)

  Implementation of MrgSorter copy constructor.
- virtual ~MrgSorter ()

  Implementation of MrgSorter destructor.
- virtual void sort ()

  Implementation of MrgSorter method to sort items with merge sort.

Private Member Functions

- void sortHelper (int leftIndex, int rightIndex)

  Implementation of MrgSorter method to assist in sorting items with merge sort.
- void mergeData (int leftIndex, int middleIndex, int rightIndex)

  Implementation of MrgSorter method to assist in sorting items with merge sort by merging the data.

Additional Inherited Members

4.3.1 Constructor & Destructor Documentation

4.3.1.1 MrgSorter::MrgSorter ( )

Implementation of MrgSorter default constructor.

The base class default constructor is called with an initializer

Precondition

Assumes an uninitialized MrgSorter object

Postcondition

An uninitialized MrgSorter object

Algorithm

The base class default constructor is called with an initializer

Exceptions

None	

Parameters

None	

Returns

None

Note

Initializer used

4.3.1.2 MrgSorter::MrgSorter (int *initialCapacity*)

Implementation of MrgSorter parameterized constructor.

The base class parameterized constructor is called with an initializer

Precondition

Assumes an uninitialized MrgSorter object

Postcondition

An initialized MrgSorter object with nodes created for the vector in the amount of the parameter passed in

Algorithm

The base class parameterized constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*initialCapacity*	An int corresponding to the number of nodes that will be created for the vector (int)

Returns

None

Note

Initializer used

4.3.1.3 MrgSorter::MrgSorter (const SorterClass< StudentType > & *copiedSorter*)

Implementation of MrgSorter copy constructor.

The base class copy constructor is called with an initializer

Precondition

Assumes an uninitialized MrgSorter object

Postcondition

A MrgSorter object with the same nodes and values as the object passed in as a parameter

Algorithm

The base class copy constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*copiedSorter*	A const referenced parameter that corresponds to the MrgSorter object to be copied into the local object (SorterClass<StudentType>)

Returns

None

Note

Initializer used

4.3.1.4 MrgSorter::∼MrgSorter ( ) `[virtual]`

Implementation of MrgSorter destructor.

Destructs the MrgSorter object

Precondition

An initialized MrgSorter object with nodes

Postcondition

All memory allocated to nodes is freed and data members are set to default values

Algorithm

The base class method resize is called with an argument of zero to clear all nodes and set data members to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.3.2 Member Function Documentation

4.3.2.1 void MrgSorter::mergeData (int *leftIndex*, int *middleIndex*, int *rightIndex*) `[private]`

Implementation of MrgSorter method to assist in sorting items with merge sort by merging the data.

Merges the data broken down by its calling method, sortHelper, into a sorted vector or subvector

Precondition

Assumes an initialized MrgSorter object with nodes which will hold objects of type StudentType

Postcondition

The vector or portions of the vector that were broken down into subvectors are reassembled in an ascending order

Algorithm

An event controlled loop goes through the two halves of the vector or subvector and puts the smallest value into a temporary StudentType array in order, then two event controlled loops check for remaining items and puts them into the temporary array, and lastly a counter controlled loop puts the sorted items from the temporary array into the vector or subvector

Exceptions

None	

Parameters

`in`	*leftIndex*	An int corresponding to the lower index of the vector or subvector (int)
`in`	*middleIndex*	An int corresponding to the middle index of the vector or subvector (int)
`in`	*rightIndex*	An int corresponding to the upper index of the vector or subvector (int)

Returns

None

Note

The items in the vector must have a compareTo method Design inspired from "" by (pg. 314)

4.3.2.2 void MrgSorter::sort ( ) `[virtual]`

Implementation of MrgSorter method to sort items with merge sort.

Calls sorterHelper to sort items in the vector with merege sort

Precondition

Assumes an initialized MrgSorter object with nodes which will hold objects of type StudentType

Postcondition

The items in the vector are sorted in ascending order

Algorithm

The method sorterHelper is called with zero and the size of the vector minus one as parameters

Exceptions

None	

Parameters

None	

Returns

None

Note

The items in the vector must have a compareTo method

Reimplemented from SorterClass< StudentType >.

4.3.2.3 void MrgSorter::sortHelper (int *leftIndex,* int *rightIndex*) `[private]`

Implementation of MrgSorter method to assist in sorting items with merge sort.

Calls itself recursively and the method mergeData to sort items in the vector with merege sort

Precondition

Assumes an initialized MrgSorter object with nodes which will hold objects of type StudentType

Postcondition

The vector or portions of the vector are broken down into subvectors and reassembled in an ascending order

Algorithm

An if statement checks that the leftmost index is less than the rightmost and if so then the middle index is calculated and then the function is called again on the lower half of the vector, then again on the upper half, which in turn call each again recursively, until popping out and calling mergeData

Exceptions

None	

Parameters

`in`	*leftIndex*	An int corresponding the the leftmost or smallest index in the vector (int)
`in`	*rightIndex*	An int corresponding the the rightmost or greatest index in the vector (int)

Returns

None

Note

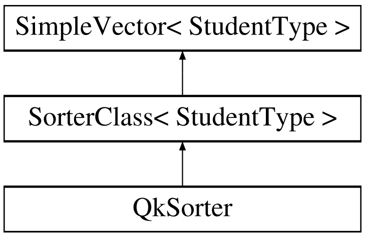
The items in the vector must have a compareTo method Design inspired from "" by (pg. 314)

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

- MrgSorter.h
- MrgSorter.cpp

4.4 QkSorter Class Reference

Inheritance diagram for QkSorter:



Public Member Functions

- QkSorter ()

    Implementation of QkSorter default constructor.
- QkSorter (int initialCapacity)

    Implementation of QkSorter parameterized constructor.
- QkSorter (const SorterClass< StudentType > &copiedSorter)

    Implementation of QkSorter copy constructor.
- virtual ∼QkSorter ()

    Implementation of QkSorter destructor.
- virtual void sort ()

    Implementation of QkSorter method to sort items with quick sort.

Private Member Functions

- void sortHelper (int leftLimitIndex, int rightLimitIndex)

    Implementation of QkSorter method to assist in sorting items with quick sort.
- int partition (int leftLimitIndex, int rightLimitIndex)

    Implementation of QkSorter method to assist in sorting items with by determining the pivot.

Additional Inherited Members

4.4.1 Constructor & Destructor Documentation

4.4.1.1 QkSorter::QkSorter ( )

Implementation of QkSorter default constructor.

The base class default constructor is called with an initializer

Precondition

Assumes an uninitialized QkSorter object

Postcondition

An uninitialized QkSorter object

Algorithm

The base class default constructor is called with an initializer

Exceptions

None	

Parameters

None	

Returns

None

Note

Initializer used

4.4.1.2 QkSorter::QkSorter (int *initialCapacity*)

Implementation of QkSorter parameterized constructor.

The base class parameterized constructor is called with an initializer

Precondition

Assumes an uninitialized QkSorter object

Postcondition

An initialized QkSorter object with nodes created for the vector in the amount of the parameter passed in

Algorithm

The base class parameterized constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*initialCapacity*	An int corresponding to the number of nodes that will be created for the vector (int)

Returns

None

Note

Initializer used

4.4.1.3 QkSorter::QkSorter (const SorterClass< StudentType > & *copiedSorter*)

Implementation of QkSorter copy constructor.

The base class copy constructor is called with an initializer

Precondition

Assumes an uninitialized QkSorter object

Postcondition

A QkSorter object with the same nodes and values as the object passed in as a parameter

Algorithm

The base class copy constructor is called with an initializer

Exceptions

None	

Parameters

`in`	*copiedSorter*	A const referenced parameter that corresponds to the QkSorter object to be copied into the local object (SorterClass<StudentType>)

Returns

None

Note

Initializer used

4.4.1.4 QkSorter::∼QkSorter ( ) `[virtual]`

Implementation of QkSorter destructor.

Destructs the QkSorter object

Precondition

An initialized QkSorter object with nodes

Postcondition

All memory allocated to nodes is freed and data members are set to default values

Algorithm

The base class method resize is called with an argument of zero to clear all nodes and set data members to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.4.2 Member Function Documentation

4.4.2.1 int QkSorter::partition (int *leftLimitIndex,* int *rightLimitIndex*) [private]

Implementation of QkSorter method to assist in sorting items with by determining the pivot.

A pivot is chosen, the lower part of the vector, then it is moved relative to the other items in the vector until it is sorted

Precondition

Assumes an initialized QkSorter object with nodes which will hold objects of type StudentType

Postcondition

One item in the vector is in its correct location, the pivot, and its index is returned

Algorithm

An event controlled loop, while the lower index is not equal to the upper index, checks whether pivot is less than the upper index or greater than the lower index, if the former then the item at pivot is compared at the upper index, they are swapped if the item at pivot is larger, otherwise the upper index is decremented, and likewise but in the opposite manner with the lower index if the latter

Exceptions

None	

Parameters

in	*leftLimitIndex*	An int corresponding to the lower index (int)
in	*rightLimitIndex*	An int corresponding to the upper index (int)

Returns

An int corresponding to the pivot, or the index which is correctly sorted (int)

Note

The items in the vector must have a compareTo method

4.4.2.2 void QkSorter::sort ( ) [virtual]

Implementation of QkSorter method to sort items with quick sort.

Calls sorterHelper to sort items in the vector with quick sort

Precondition

Assumes an initialized QkSorter object with nodes which will hold objects of type StudentType

Postcondition

The items in the vector are sorted in ascending order

Algorithm

The method sorterHelper is called with zero and the size of the vector minus one as parameters

Exceptions

None	

Parameters

None	

Returns

None

Note

The items in the vector must have a compareTo method

Reimplemented from SorterClass< StudentType >.

4.4.2.3 void QkSorter::sortHelper (int *leftLimitIndex,* int *rightLimitIndex*) `[private]`

Implementation of QkSorter method to assist in sorting items with quick sort.

Calls the method partition and itself recursively to sort items in the vector with quick sort

Precondition

Assumes an initialized QkSorter object with nodes which will hold objects of type StudentType

Postcondition

A pivot index is determined with all items smaller to the left of it and all items larger to the right and then the method is called recursively on the two sides of the pivot with the lower and upper halves of the vector passed in as indices

Algorithm

An if statement checks that the lower index is less than the upper index and if so then the method partition is called to get an index corresponding to the pivot and then the vector halves below and above that pivot are sorted recursively with additional calls to sortHelper

Exceptions

None	

Parameters

`in`	*leftLimitIndex*	An int corresponding to the lower index of the vector (int)
`in`	*rightLimitIndex*	An int corresponding to the upper index of the vector (int)

Returns

None

Note

The items in the vector must have a compareTo method

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

- QkSorter.h
- QkSorter.cpp

4.5 SimpleTimer Class Reference

Public Member Functions

- SimpleTimer ()
  Default constructor.
- ∼SimpleTimer ()
  Default constructor.
- void start ()
  Start control.
- void stop ()
  Stop control.
- void **getElapsedTime** (char ∗timeStr)

Static Public Attributes

- static const char **NULL_CHAR** = '\0'
- static const char **RADIX_POINT** = '.'

Private Attributes

- struct timeval startData **endData**
- long int **beginTime**
- long int **endTime**
- long int **secTime**
- long int **microSecTime**
- bool **running**
- bool **dataGood**

4.5.1 Constructor & Destructor Documentation

4.5.1.1 SimpleTimer::SimpleTimer ( )

Default constructor.

Constructs Timer class

Parameters

None	

Note

set running flag to false

4.5.1.2 SimpleTimer::∼SimpleTimer ( )

Default constructor.

Destructs Timer class

Parameters

None	

Note

No data to clear

4.5.2 Member Function Documentation

4.5.2.1 void SimpleTimer::start ( )

Start control.

Takes initial time data

Parameters

None	

Note

None

4.5.2.2 void SimpleTimer::stop ( )

Stop control.

Takes final time data, calculates duration

Parameters

None	

Note

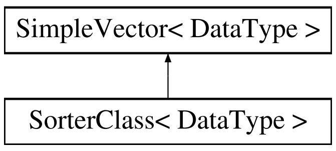
None

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

- SimpleTimer.h
- SimpleTimer.cpp

4.6 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**

4.6.1 Constructor & Destructor Documentation

4.6.1.1 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

4.6.1.2 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

4.6.1.3 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

4.6.1.4 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

4.6.2 Member Function Documentation

4.6.2.1 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

4.6.2.2 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

4.6.2.3 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

4.6.2.4 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

4.6.2.5 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

4.6.2.6 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

4.6.2.7 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

4.6.2.8 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

4.6.2.9 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

4.6.2.10 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

4.6.2.11 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

4.6.2.12 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

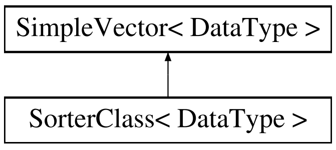
Provided as convenience for user; has no impact on SimpleVector data

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

- SimpleVector.h
- SimpleVector.cpp

4.7 SorterClass< DataType > Class Template Reference

Inheritance diagram for SorterClass< DataType >:



Public Member Functions

- SorterClass (int newCapacity=DEFAULT_CAPACITY)

    Implementation of SorterClass default constructor.
- SorterClass (int newCapacity, const DataType &fillValue)

    Implementation of SorterClass parameterized constructor.
- SorterClass (const SorterClass< DataType > &copiedVector)

    Implementation of SorterClass copy constructor.
- virtual ~SorterClass ()

    Implementation of SorterClass destructor.
- void add (const DataType &addedItem)

    Implementation of SorterClass method to add an item to the vector.
- bool remove (DataType &removedItem)

    Implementation of SorterClass method to remove an item from the vector.
- int findIndexFor (const DataType &searchItem)

    Implementation of SorterClass method to find an item in the vector.
- virtual void sort ()

    Virtual SorterClass method to sort items in vector.
- void copyFromTo (int indexTo, int indexFrom)

    Implementation of SorterClass method to copy and item from one index to another.
- void swapBetween (int oneIndex, int otherIndex)

    Implementation of SorterClass method to swap items at two indices.
- void insertAtIndex (int insertIndex, const DataType &itemToInsert)

    Implementation of SorterClass method to insert an item in the vector.
- void removeAtIndex (int removalIndex, DataType &removedItem)

    Implementation of SorterClass method to remove an item from the vector.

Static Public Attributes

- static const int **DEFAULT_CAPACITY** = 10
- static const int **NOT_FOUND** = -1

4.7.1 Constructor & Destructor Documentation

4.7.1.1 template<class DataType > SorterClass< DataType >::SorterClass (int *newCapacity* = DEFAULT_CAPACITY)

Implementation of SorterClass default constructor.

Default constructor of base class constructs the object

Precondition

Assumes an uninitialized SorterClass object

Postcondition

The object's vector has nodes created for it in the amount of the parameter passed in

Algorithm

The default constructor of the base class, SimpleVector, is invoked with an initializer

Exceptions

None	

Parameters

`in`	*newCapacity*	An int that corresponds to the capacity of the vector (int)

Returns

None

Note

Initializer used

4.7.1.2 template<class DataType> SorterClass< DataType >::SorterClass (int *newCapacity,* const DataType & *fillValue*)

Implementation of SorterClass parameterized constructor.

Parameterized constructor of base class constructs the object

Precondition

Assumes an uninitialized SorterClass object

Postcondition

The object's vector has nodes created for it and filled by the parameters passed in

Algorithm

The parameterized constructor of the base class, SimpleVector, is invoked with an initializer

Exceptions

None	

Parameters

`in`	*newCapacity*	An int that corresponds to the capacity of the vector (int)
`in`	*fillValue*	A const reference parameter of type DataType which will be used to fill the vector (DataType)

Returns

None

Note

Initializer used

4.7.1.3 template<class DataType> SorterClass< DataType >::SorterClass (const SorterClass< DataType > & *copiedVector*)

Implementation of SorterClass copy constructor.

Copy constructor of base class constructs the object

Precondition

Assumes an uninitialized SorterClass object

Postcondition

The object's vector has nodes created for it and filled with the same values as those of the vector passed in as a parameter

Algorithm

The copy constructor of the base class, SimpleVector, is invoked with an initializer

Exceptions

None	

Parameters

`in`	*copiedVector*	A reference parameter of type SorterClass that will have its values copied into the calling object (SorterClass<DataType>)

Returns

None

Note

Initializer used

4.7.1.4 template<class DataType > SorterClass< DataType >::~SorterClass ( ) `[virtual]`

Implementation of SorterClass destructor.

The base class method resize deletes all vector nodes

Precondition

Assumes an initialized SorterClass object

Postcondition

All nodes in the vector are deleted and data members are set to default values

Algorithm

The base class method resize is called with an argument of zero to clear the vector and set data members to default values

Exceptions

None	

Parameters

None	

Returns

None

Note

None

4.7.2 Member Function Documentation

4.7.2.1 template<class DataType> void SorterClass< DataType >::add (const DataType & *addedItem*)

Implementation of SorterClass method to add an item to the vector.

The item passed in as parameter is added to the end of the vector

Precondition

Assumes an initialized SorterClass object

Postcondition

The SorterClass object has the item passed in as parameter added to the vector

Algorithm

An if statement checks whether the vector is full and if so more nodes are created with resize and then the method insertAtIndex is called to insert the item

Exceptions

Base	class method setAtIndex is indirectly invoked which throws a logic error if an index given to it is out of bounds

Parameters

`in`	*addedItem*	A const reference parameter of type DataType which corresponds to the item to be added to the vector (DataType)

Returns

None

Note

None

4.7.2.2 template<class DataType > void SorterClass< DataType >::copyFromTo (int *indexTo,* int *indexFrom*)

Implementation of SorterClass method to copy and item from one index to another.

The item at one index is copied into another

Precondition

Assumes an initialized SorterClass object with nodes

Postcondition

The item at one index specified by one of the parameters is copied into the index specified by the other parameter

Algorithm

An if statement checks whether the parameters are within the bounds of the vector size and if there are nodes and if so the base class methods getAtIndex and setAtIndex get the item from one index and set it at the other

Exceptions

Base	class method setAtIndex is invoked which throws a logic error if an index given to it is out of bounds

Parameters

`in`	*indexTo*	An int corresponding to the index to which the item will be copied (int)
`in`	*indexFrom*	An int corresponding to the index from which the item will be copied (int)

Returns

None

Note

None

4.7.2.3 template<class DataType> int SorterClass< DataType >::findIndexFor (const DataType & *searchItem*)

Implementation of SorterClass method to find an item in the vector.

Takes in an item of type DataType and searches for it in the vector

Precondition

Assumes an initialized SorterClass object with nodes

Postcondition

The item passed in as a parameter is searched for in the vector and and the SorterClass object is unchanged

Algorithm

An if statement checks that there are nodes and then if so a counter controlled loop moves through the vector and each item is compared with the DataType method compareTo and if there is a match then the index is returned, otherwise NOT_FOUND (-1) is returned

Exceptions

None	

Parameters

in	*searchItem*	A const reference parameter of type DataType which is to be searched for in the vector (DataType)

Returns

An int corresponding to the index at which the item passed in as a parameter was found, NOT_FOUND (-1) is returned if it's not found (int)

Note

Assumes that DataType has a compareTo method

4.7.2.4 template<class DataType> void SorterClass< DataType >::insertAtIndex (int *insertIndex,* const DataType & *itemToInsert*)

Implementation of SorterClass method to insert an item in the vector.

The index given as a parameter specifies where the DataType item is to be inserted and subsequent items are shifted down

Precondition

Assumes an initialized SorterClass object with nodes and a valid index parameter

Postcondition

The vector contains the item inserted at the index specified and the items after it are shifted down

Algorithm

An if statement checks that the index is valid and that there's room to shift items down and if so then if size is greater than zero then a counter controlled loop shifts everything down one, then the method setAtIndex inserts the item and the size is incremented

Exceptions

Base	class method setAtIndex is invoked which throws a logic error if an index given to it is out of bounds

Parameters

in	*insertIndex*	An int corresponding to the index at which the item is to be inserted (int)
in	*itemToInsert*	A const referenced parameter of type DataType corresponding to the item to be inserted (DataType)

Returns

None

Note

None

4.7.2.5 template<class DataType> bool SorterClass< DataType >::remove (DataType & *removedItem*)

Implementation of SorterClass method to remove an item from the vector.

The item passed in as parameter is searched fro in the vector and if it's there it is removed and stored in the parameter

Precondition

Assumes an initialized SorterClass object with nodes and a size greater than zero

Postcondition

The item passed in as parameter is removed from the vector if it's found and then that item is stored in the parameter

Algorithm

An if statement checks for nodes and then the method findAtIndex is called to find the item in the vector, if it's not found then false is returned, if it is then removeAtIndex is called to remove it

Exceptions

Base	class method removeAtIndex is indirectly invoked which throws a logic error if an index given to it is out of bounds

Parameters

`out`	*removedItem*	A reference parameter of type DataType that is to be searched for in the vector and removed and then stored in the parameter (DataType)

Returns

A bool corresponding to whether or not there are nodes with values and whether or not the item was found (bool)

Note

None

4.7.2.6 template<class DataType> void SorterClass< DataType >::removeAtIndex (int *removalIndex,* DataType & *removedItem*)

Implementation of SorterClass method to remove an item from the vector.

The index given as a parameter specifies the location from which the DataType item is to be removed and subsequent items are shifted up

Precondition

Assumes an initialized SorterClass object with nodes and a valid index parameter

Postcondition

The vector has the item removed at the index specified, the items after it are shifted up and then DataType parameter stores the item removed

Algorithm

An if statement checks that the index is valid and if it is then the DataType parameter receives the item at that index with a call to getAtIndex, then an if statement checks that there are more than one node, if so a counter controlled loop shifts everything in the vector up one and size is decremented, otherwise zeroSize is called

Exceptions

None	

Parameters

`in`	*removalIndex*	An int corresponding to the index at which the item is to be removed (int)
`out`	*removedItem*	A const referenced parameter of type DataType corresponding to the item to be removed which will store the removed item (DataType)

Returns

None

Note

None

4.7.2.7 template<class DataType > void SorterClass< DataType >::sort ( ) `[virtual]`

Virtual SorterClass method to sort items in vector.

Leaves implementation to derived classes to sort items in the vector

Precondition

Assumes an initialized class object inherited from SorterClass containing nodes

Postcondition

The items in the vector of the derived classed are sorted

Algorithm

As a virtual method, derived classes will implement the method

Exceptions

None	

Parameters

None	

Returns

None

Note

Derived classes will implement this method

Reimplemented in QkSorter, BblSorter, and MrgSorter.

4.7.2.8 template<class DataType > void SorterClass< DataType >::swapBetween (int *oneIndex,* int *otherIndex*)

Implementation of SorterClass method to swap items at two indices.

The items at the two indices specified by the parameters are swapped

Precondition

Assumes an initialized SorterClass object with nodes and valid index parameters

Postcondition

The items at the indices specified by the parameters are swapped

Algorithm

An if statement checks that there are nodes and that the indices are and if so then the base class method getAtIndex gets the item at the first index, stores it in a temporary DataType variable, then uses the base class method setAtIndex to set the item at the other index into the first and then puts the item in temp into the other index

Exceptions

Base	class method setAtIndex is invoked which throws a logic error if an index given to it is out of bounds

Parameters

`in`	*oneIndex*	An int corresponding to an index at which an item is to be swapped (int)
`in`	*otherIndex*	An int corresponding to the other index at which an item is to be swapped (int)

Returns

None

Note

None

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

- SorterClass.h
- SorterClass.cpp

4.8 StudentType Class Reference

Public Member Functions

- StudentType ()

  Default/Initialization constructor.
- StudentType (char ∗initStudentName, int initUnivIDNum, char initGender)

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

  Assignment operation.
- void setStudentData (char ∗inStudentName, int inStudentID, char inGender)

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

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

  Data serialization.

Static Public Attributes

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

Private Member Functions

- void copyString (char ∗destination, const char ∗source) const

  String copy utility.
- void parseNames (char ∗lastName, char ∗firstName, const char ∗fullName) const

  Name parsing utility.
- int compareStrings (const char ∗oneStr, const char ∗otherStr) const

  String comparison facility.
- char toLower (char testChar) const

  Letter to lower case facility.

Private Attributes

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

4.8.1 Constructor & Destructor Documentation

4.8.1.1 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

4.8.1.2 StudentType::StudentType (char $*$ *initStudentName,* int *initUnivIDNum,* char *initGender*)

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 gender

Exceptions

None	

Parameters

`in`	*initStudentName*	Name of student as c-string
`in`	*initUnivIDNum*	University ID number as integer
`in`	*initGender*	gender

Returns

None

Note

None

4.8.2 Member Function Documentation

4.8.2.1 int StudentType::compareStrings (const char $*$ *oneStr,* const char $*$ *otherStr*) const `[private]`

String comparison facility.

Compares two strings ignoring case

Precondition

assumes standard string conditions, including NULL_CHAR end

Postcondition

first name and last name strings hold correct components of original full name string

Algorithm

Compares letters one by one with each letter set to lower case, if a difference in letter is found, it is returned, if the end of the shortest string is reached without a difference, strings are assumed to be the same Returns 0 if strings are equal, returns $>$ 0 if one string $>$ other string returns $<$ 0 if one string $<$ other string

Exceptions

None	

Parameters

`in`	*oneStr*	One of the two strings to be compared
`in`	*otherStr*	The other of the two strings to be compared

Returns

Difference between two strings (int)

Note

None

4.8.2.2 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

Parses student name into last and first using parseName, then returns test for last name first, then first name

Exceptions

None	

Parameters

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

Returns

Integer result of comparison process

Note

None

4.8.2.3 void StudentType::copyString (char ∗ *destination,* const char ∗ *source*) const [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

4.8.2.4 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

4.8.2.5 void StudentType::parseNames (char ∗ *lastName,* char ∗ *firstName,* const char ∗ *fullName*) const [private]

Name parsing utility.

Takes full name and breaks into first and last names

Precondition

assumes standard string conditions, including NULL_CHAR end

Postcondition

first name and last name strings hold correct components of original full name string

Algorithm

Copies string character by character from source into last name string until a comma is found, then it copies the remainder into the first name string, assumes string max length $<$ STD_STR_LEN

Exceptions

None	

Parameters

out	*lastName*	String containing last name of student
out	*firstName*	String containing first name of student
in	*fullName*	String containing full name of student, with first and last names delimited by a comma

Returns

None

Note

None

4.8.2.6 void StudentType::setStudentData (char ∗ *inStudentName,* int *inStudentID,* char *inGender*)

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`	*gender*	Character identifier for gender

Returns

Integer result of comparison process

Note

None

4.8.2.7 char StudentType::toLower (char *testChar*) const `[private]`

Letter to lower case facility.

None

Precondition

No assumptions are made related to the input data

Postcondition

If the character is an upper case letter, it is converted to lower case and returned; otherwise the character is returned unchanged

Algorithm

Tests for upper case letter; If upper case, letter is mathematically modifed to lower case otherwise no action is taken

Exceptions

None	

Parameters

`in`	*testChar*	Character to be tested for upper case and modified as needed

Returns

None

Note

None

4.8.2.8 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

5 File Documentation

5.1 BblSorter.cpp File Reference

Implementation file for BblSorter class.

```
#include "BblSorter.h"
#include "StudentType.h"
#include "SorterClass.cpp"
```

5.1.1 Detailed Description

Implementation file for BblSorter class.

Author

Bryan Kline

Implements all member methods for BblSorter class

Version

1.00 Bryan Kline (02/24/16)

None

5.2 BblSorter.h File Reference

Definition file for BblSorter class using insertion sort, derived from SorterClass.

```
#include "StudentType.h"
#include "SorterClass.cpp"
```

Classes

- class BblSorter

5.2.1 Detailed Description

Definition file for BblSorter class using insertion sort, derived from SorterClass. Specifies all member methods of the BblSorter Class

Version

1.10 Michael Leverington (12 February 2016) Updated for use with new SorterClass

1.00 Michael Leverington (19 September 2015) Original code

Requires StudentType.h, SorterClass.h

5.3 MrgSorter.cpp File Reference

Implementation file for MrgSorter class.

```
#include "MrgSorter.h"
#include "StudentType.h"
#include "SorterClass.cpp"
```

Variables

- static const int **TWO** = 2

5.3.1 Detailed Description

Implementation file for MrgSorter class.

Author

Bryan Kline

Implements all member methods for MrgSorter class

Version

1.00 Bryan Kline (02/24/16)

None

5.4 MrgSorter.h File Reference

Definition file for MrgSorter class using insertion sort, derived from SorterClass.

```
#include "StudentType.h"
#include "SorterClass.h"
```

Classes

- class MrgSorter

5.4.1 Detailed Description

Definition file for MrgSorter class using insertion sort, derived from SorterClass. Specifies all member methods of the MrgSorter Class

Version

1.10 Michael Leverington (12 February 2016) Updated for use with new SorterClass

1.00 Michael Leverington (19 September 2015) Original code

Requires StudentType.h, SorterClass.h

5.5 QkSorter.cpp File Reference

Implementation file for QkSorter class.

```
#include "QkSorter.h"
#include "StudentType.h"
#include "SorterClass.cpp"
```

5.5.1 Detailed Description

Implementation file for QkSorter class.

Author

Bryan Kline

Implements all member methods for QkSorter class

Version

1.00 Bryan Kline (02/24/16)

None

5.6 QkSorter.h File Reference

Definition file for QkSorter class using insertion sort, derived from SorterClass.

```
#include "StudentType.h"
#include "SorterClass.h"
```

Classes

- class QkSorter

5.6.1 Detailed Description

Definition file for QkSorter class using insertion sort, derived from SorterClass.

Author

Michael Leverington

Specifies all member methods of the QkSorter Class

Version

1.10 Michael Leverington (12 February 2016) Updated for use with new SorterClass

1.00 Michael Leverington (19 September 2015) Original code

Requires StudentType.h, SorterClass.h

5.7 SimpleTimer.cpp File Reference

Implementation file for SimpleTimer class.

```
#include "SimpleTimer.h"
```

5.7.1 Detailed Description

Implementation file for SimpleTimer class.

Author

Michael Leverington

Implements member methods for timing

Version

1.00 (11 September 2015)

Requires SimpleTimer.h.

5.8 SimpleTimer.h File Reference

Definition file for simple timer class.

```
#include <sys/time.h>
#include <cstring>
```

Classes

- class SimpleTimer

5.8.1 Detailed Description

Definition file for simple timer class.

Author

Michael Leverington

Specifies all member methods of the SimpleTimer

Version

1.00 (11 September 2015)

None

5.9 SimpleVector.cpp File Reference

Implementation file for SimpleVector class.

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

5.9.1 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

5.10 SimpleVector.h File Reference

Definition file for SimpleVector class.

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

Classes

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

5.10.1 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

5.11 SorterClass.cpp File Reference

Implementation file for SorterClass class.

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

5.11.1 Detailed Description

Implementation file for SorterClass class.

Author

Bryan Kline

Implements all member methods for SorterClass class

Version

1.00 Bryan Kline (02/21/2016)

None

5.12 SorterClass.h File Reference

Definition file for SorterClass class.

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

Classes

- class SorterClass< DataType >

5.12.1 Detailed Description

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

Version

1.00 Michael Leverington (29 January 2016) Original code

Requires SimpleVector.h

5.13 StudentType.cpp File Reference

Implementation file for StudentType class.

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

5.13.1 Detailed Description

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

Version

1.10 Michael Leverington (10 February 2016) Update for use with SorterClass

1.00 Michael Leverington (30 January 2016) Initial development

Requires StudentType.h

Index