PA02 - SimpleVector

Generated by Doxygen 1.8.6

Tue Feb 2 2016 15:28:39

Contents

1 Class Index

1.1 Class List

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

DataNode< DataType >	**2**
SimpleVector< DataType >	**3**

2 File Index

2.1 File List

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

PA02.cpp
Driver program to exercise linked-list based Vector classes **14**

SimpleVector.cpp
Implementation file for SimpleVector and DataNode classes **15**

3 Class Documentation

3.1 DataNode< DataType > Class Template Reference

Public Member Functions

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

  Implementation of templated DataNode parameterized constructor.

Public Attributes

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

3.1.1 Constructor & Destructor Documentation

3.1.1.1 template<typename DataType > DataNode< DataType >::DataNode (DataType & *inData,* DataNode< DataType > ∗ *inPrevPtr,* DataNode< DataType > ∗ *inNextPtr*)

Implementation of templated DataNode parameterized constructor.

The parameters passed into the constructor are assigned to the data members of the DataNode object

Precondition

An uninitialized DataNode object

Postcondition

A DataNode object with data members initialized to the values passed passed in as parameters

Algorithm

Initializers are used to set data members to the values passed in as parameters

Exceptions

None	

Parameters

`in`	*inData*	Parameter of type DataType passed by reference into the template which will be the data value that the node holds inPrevPtr (<DataType>) Parameter of type DataNode pointer which points to the previous node in the vector (DataNode<DataType>) inNextPtr Parameter of type DataNode pointer which points to the next node in the vector (DataNode<DataType>)

Returns

None

Note

Initializers used

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

- SimpleVector.h
- SimpleVector.cpp

3.2 SimpleVector< DataType > Class Template Reference

Public Member Functions

- SimpleVector (int newCapacity=DEFAULT_CAPACITY)

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

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

  Implementation of templated SimpleVector copy constructor.
- ∼SimpleVector ()

  Implementation of templated SimpleVector destructor.
- const SimpleVector< DataType > & operator= (const SimpleVector< DataType > &rhVector)

  Implementation of templated SimpleVector overloaded assignment operator.
- int getCapacity () const

  Implementation of templated SimpleVector method which returns the capacity of the vector.
- int getSize () const

  Implementation of templated SimpleVector method which returns the size of the vector.
- void showStructure (char IDChar) const

  Implementation of templated SimpleVector method which prints the vector list nodes to the screen.
- void setAtIndex (int index, const DataType &inData) throw (logic_error)

  Implementation of templated SimpleVector method which sets an item at a given index in the vector.
- const DataType & getAtIndex (int index) throw (logic_error)

  Implementation of templated SimpleVector method which gets an item at a given index from the vector.
- void resize (int newCapacity)

  Implementation of templated SimpleVector method which changes the capacity of the vector.
- void incrementSize ()

  Implementation of templated SimpleVector method to increment the size of the vector.
- void decrementSize ()

  Implementation of templated SimpleVector method to decrement vector size.
- void zeroSize ()

  Implementation of templated SimpleVector method to set vector size to zero.

Static Public Attributes

- 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)

  Implementation of private templated SimpleVector method which copies a SimpleVector object into the calling Simple-Vector object.
- DataNode< DataType > ∗ getPointerToIndex (int index)

  Implementation of private templated SimpleVector method to return a pointer to a node in the vector.

Private Attributes

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

3.2.1 Constructor & Destructor Documentation

3.2.1.1 template<typename DataType > SimpleVector< DataType >::SimpleVector (int *newCapacity =* `DEFAULT_CAPACITY`)

Implementation of templated SimpleVector default constructor.

Initializers set default values to the data members in the vector and the vector is resized to the capacity passed in as a parameter

Precondition

An uninitialized SimpleVector object

Postcondition

An initialized SimpleVector object with default values and nodes created in the amount of the parameter new-Capacity

Algorithm

Initializers are used to set data members to default values and the method resize is called with newCapacity as an argument to create nodes for the vector

Exceptions

None	

Parameters

`in`	*newCapacity*	An int which initializes vector capacity, or the maximum number of nodes the vector can contain (int)

Returns

None

Note

Initializers used

3.2.1.2 template<typename DataType > SimpleVector< DataType >::SimpleVector (int *newCapacity*, const DataType & *fillValue*)

Implementation of templated SimpleVector parameterized constructor.

A SimpleVector is created and filled with the value or object of type DataType passed in as a parameter

Precondition

An uninitialized SimpleVector object

Postcondition

A SimpleVector object with newCapacity number of nodes, all of which filled with fillValue of type DataType

Algorithm

Initializers are used to set data members to default values, the method resize is called with newCapacity as an argument to change the capacity of the vector, and then a counter controlled loop moves through the vector and the method setAtIndex is called with the parameter fillValue passed into it as the value to fill the vector

Exceptions

None	

Parameters

in	*newCapacity*	An int which initializes vector capacity, or the maximum number of nodes the vector can contain (int) fillValue A reference parameter of type DataType which will be the value or object which is used to fill all the nodes in the vector (<-DataType>)

Returns

None

Note

Initializers used

3.2.1.3 template<typename DataType > SimpleVector< DataType >::SimpleVector (const SimpleVector< DataType > & *copiedVector*)

Implementation of templated SimpleVector copy constructor.

The SimpleVector object passed into the constructor as a parameter is copied into the SimpleVector object to be constructed

Precondition

An uninitialized SimpleVector object

Postcondition

A SimpleVector object with the same nodes and data member values as the object passed in as a parameter

Algorithm

The method copyVectorObject is called with the parameter copiedVector passed in as an argument

Exceptions

None	

Parameters

`in`	*copiedVector*	A const SimpleVector object reference parameter which has its nodes and data values copied into the constructing SimpleVector object (SimpleVector<Data-Type>)

Returns

None

Note

None

3.2.1.4 template<typename DataType > SimpleVector< DataType >::~SimpleVector ( )

Implementation of templated SimpleVector destructor.

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

Precondition

An initialized SimpleVector object

Postcondition

All memory allocated for nodes in the vector freed and data members set to defaul values

Algorithm

The method resize is called with zero passed in to clear out the vector and the method zeroSize is called to set vectorSize to zero

Exceptions

None	

Parameters

None	

Returns

None

Note

None

3.2.2 Member Function Documentation

3.2.2.1 template<typename DataType > void SimpleVector< DataType >::copyVectorObject (const SimpleVector< DataType > & *inData*) `[private]`

Implementation of private templated SimpleVector method which copies a SimpleVector object into the calling SimpleVector object.

The SimpleVector object passed into the method as a parameter is copied into the calling SimpleVector object

Precondition

An SimpleVector object

Postcondition

The calling SimpleVector object has the same nodes and data member values as the object passed in as a parameter

Algorithm

The overloaded assignment operator is called on the parameter inData and this dereferenced

Exceptions

None	

Parameters

`in`	*inData*	A const SimpleVector object reference parameter which will be copied into the calling vector (SimpleVector<DataType>)

Returns

None

Note

Method is private

3.2.2.2 template<typename DataType > void SimpleVector< DataType >::decrementSize ( )

Implementation of templated SimpleVector method to decrement vector size.

The vector data member vectorSize is decreased by one

Precondition

An initialized SimpleVector object

Postcondition

The data member vectorSize is changed

Algorithm

An if statement checks whether the data member vectorSize is greater than zero and if so it's decremented

Exceptions

None	

Parameters

None	

Returns

None

Note

Method intended for programmer convenience

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

Implementation of templated SimpleVector method which gets an item at a given index from the vector.

The data portion of the node at the location in the vector as specified by the parameter index is returned from the method

Precondition

An initialized SimpleVector object containing at least one node

Postcondition

The value or object in the data portion of the node at the index in the SimpleVector object specified by the parameter index is returned and the vector is unchanged

Algorithm

An if statement checks if index is valid, if not then an exception is thrown, if so then the method getPointerTo-Index is called with index as a parameter and that is assigned to currentPtr, then currentIndex is updated and the value or object in the node is returned

Exceptions

If	the parameter index is less than zero or greater than vectorCapacity, meaning past the end of the vector, then a logic_error is thrown returning the string "Error/: invalid index"

Parameters

`in`	*index*	An int corresponding to the index from which an item should be returned from the vector (int)

Returns

The value or object in the data portion of the DataNode is returned (DataType)

Note

None

3.2.2.4 template<typename DataType > int SimpleVector< DataType >::getCapacity ( ) const

Implementation of templated SimpleVector method which returns the capacity of the vector.

The data member vectorCapacity is returned

Precondition

An initialized SimpleVector object

Postcondition

The SimpleVector object is unchanged and vectorCapacity is returned

Algorithm

A return statement returns the data member vectorCapacity

Exceptions

None	

Parameters

None	

Returns

An int is returned which corresponds to the capacity of the vector (int)

Note

None

3.2.2.5 template<typename DataType > DataNode< DataType > ∗ SimpleVector< DataType >::getPointerToIndex (int *index*) [private]

Implementation of private templated SimpleVector method to return a pointer to a node in the vector.

Private method which, if the parameter index is valid, moves through the vector and when it reaches that index it returns a pointer to the node at that index

Precondition

An initialized SimpleVector object

Postcondition

A pointer to the node at the index specified by the parameter index if it exists, otherwise a pointer set to NULL, is returned and the vector is unchanged

Algorithm

If statements check that the parameter index is valid, if it is then a pointer is moved to that index with a counter controlled loop, otherwise the pointer set to NULL, and the pointer is returned

Exceptions

None	

Parameters

in	*index*	An int which corresponds to the index in the vector to which a pointer should be returned if a node there exists (int)

Returns

A pointer to the node at index in the vector

Note

Method is private, and it returns a pointer set to NULL if conditions are not met (DataNode<DataType>∗)

3.2.2.6 template<typename DataType > int SimpleVector< DataType >::getSize ( ) const

Implementation of templated SimpleVector method which returns the size of the vector.

The data member vectorSize is returned

Precondition

An initialized SimpleVector object

Postcondition

The SimpleVector object is unchanged and the vector size is returned

Algorithm

A return statement returns the data member vectorSize

Exceptions

None	

Parameters

None	

Returns

An int is returned which corresponds to the size of the vector (int)

Note

None

3.2.2.7 template<typename DataType > void SimpleVector< DataType >::incrementSize ( )

Implementation of templated SimpleVector method to increment the size of the vector.

The vector data member vectorSize is increased by one

Precondition

An initialized SimpleVector object

Postcondition

The data member vectorSize is changed

Algorithm

An if statement checks whether the data member vectorSize is less than vectorCapacity and if so it's incremented

Exceptions

None	

Parameters

None	

Returns

None

Note

Method intended for programmer convenience

3.2.2.8 template<typename DataType > const SimpleVector< DataType > & SimpleVector< DataType >::operator= (const SimpleVector< DataType > & *rhVector*)

Implementation of templated SimpleVector overloaded assignment operator.

The SimpleVector object passed into the method as a parameter is copied into the calling SimpleVector object

Precondition

An SimpleVector object

Postcondition

The calling SimpleVector object has the same nodes and data member values as the SimpleVector object passed in as a parameter

Algorithm

An if statement checks whether both objects are the same, if they're not then the calling object is resized with vectorCapacity of rhVector passed in as an argument, a counter controlled loop moves through rhVector with a temporary DataNode pointer and the calling object copies in the values or objects from rhVector with a call to setAtIndex, data members are assigned the values from rhVector and and this dereferenced is returned

Exceptions

None	

Parameters

`in`	*rhVector*	A const SimpleVector object reference parameter which will be copied into the calling vector (SimpleVector<DataType>)

Returns

The calling SimpleVector object is returned with this dereferenced (SimpleVector<DataType>)

Note

None

3.2.2.9 template<typename DataType > void SimpleVector< DataType >::resize (int *newCapacity*)

Implementation of templated SimpleVector method which changes the capacity of the vector.

The nodes in the vector are either created or destroyed depending on the parameter newCapacity so that the total capacity of the vector is changed

Precondition

An initialized SimpleVector object

Postcondition

A SimpleVector object with its capacity changed

Algorithm

An if statement checks whether newCapacity is greater than zero and not equal to vectorCapacity, if so then if newCapacity is less than vectorCapacity then a temporary node pointer goes to the end of the vector and then deletes back through the vector, moving currentIndex if necessary, and if newCapacity is not less than vectorCapacity then if listHead is NULL then it is created, otherwise the appropriate number of nodes is created starting at the end of the vector, and finally vectorCapacity is updated

Exceptions

None	

Parameters

`in`	*newCapacity*	An int which specifies the new capacity of the vector (int)

Returns

None

Note

None

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

Implementation of templated SimpleVector method which sets an item at a given index in the vector.

The data portion of the DataNode at the location in the vector as specified by the parameter index is assigned a value or object corresponding to the parameter inData

Precondition

An initialized SimpleVector object containing at least one node

Postcondition

The value at the node at the index in the SimpleVector object specified by the parameter index is set to the parameter inData

Algorithm

An if statement checks if index is valid, if not then an exception is thrown, if so then the method getPointerTo-Index is called with index as a parameter and that is assigned to currentPtr, then currentPtr has the data item of the node it points to set to inData, and currentIndex is updated

Exceptions

If	the parameter index is less than zero or greater than vectorCapacity, meaning past the end of the vector, then a logic_error is thrown returning the string "Error/: invalid index"

Parameters

`in`	*index*	An int corresponding to the index at which an item should be added to the vector (int) inData A const reference parameter of type DataType which will be added to the vector at the specified index (<DataType>)

Returns

None

Note

None

3.2.2.11 template<typename DataType > void SimpleVector< DataType >::showStructure (char *IDChar*) const

Implementation of templated SimpleVector method which prints the vector list nodes to the screen.

If the vector contains any nodes then they, along with the list identifier IDChar, are printed to the screen in rows of five, otherwise it is indicated that the vector is empty

Precondition

A SimpleVector object

Postcondition

The SimpleVector object is unchanged and its contents printed to the screen

Algorithm

An if statement checks whether the vector is empty, if it is then an indication of that is printed to the screen, otherwise a counter controlled loop moves a temporary DataNode pointer through the vector, printing each one to the screen as it goes, printing a new line and spaces out every five nodes for proper formatting

Exceptions

None	

Parameters

`in`	*IDChar*	A char which acts as an identifier for the vector being printed to the screen (char)

Returns

None

Note

None

3.2.2.12 template<typename DataType > void SimpleVector< DataType >::zeroSize ( )

Implementation of templated SimpleVector method to set vector size to zero.

The vector data member vectorSize is set to zero

Precondition

An initialized SimpleVector object

Postcondition

The data member vectorSize is changed

Algorithm

The data member vectorSize is set to zero

Exceptions

None	

Parameters

None	

Returns

None

Note

Method intended for programmer convenience

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

- SimpleVector.h
- SimpleVector.cpp

4 File Documentation

4.1 PA02.cpp File Reference

Driver program to exercise linked-list based Vector classes.

```
#include <iostream>
#include <cstring>
#include "SimpleVector.cpp"
```

Functions

- void ShowMenu ()

  Displays choice of commands for exercising linked list.
- int **main** ()

Variables

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

4.1.1 Detailed Description

Driver program to exercise linked-list based Vector classes. Allows for testing all SimpleVector methods in an interactive operation

Version

1.00 Original development (23 January 2016)

Note

Requires SimpleVector.h, SimpleVector.cpp

4.1.2 Function Documentation

4.1.2.1 void ShowMenu ( )

Displays choice of commands for exercising linked list.

Command letters displayed are unique characters specified as shown

Precondition

None

Postcondition

Choice of commands is displayed as specified

Algorithm

Standard output operations for each command line available

Exceptions

None	

Parameters

None	

Returns

None

Note

Five spaces for parameter parentheses, three spaces for curly braces

4.2 SimpleVector.cpp File Reference

Implementation file for SimpleVector and DataNode classes.

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

Variables

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

4.2.1 Detailed Description

Implementation file for SimpleVector and DataNode classes. Implements member methods of SimpleVector and DataNode classes

Version

2.00 Bryan Kline (02 February 2016)

Note

Requires SimpleVector.h

4.3 SimpleVector.h File Reference

Definition file for SimpleVector class.

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

Classes

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

4.3.1 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

Index